Graph Kernels for RDF Data

Uta Lösch[1], Stephan Bloehdorn[2], and Achim Rettinger[1]

[1] Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany
uta.loesch@kit.edu, rettinger@kit.edu
[2] IBM Germany, 12277 Berlin, Germany
bloehdorn@de.ibm.com

Abstract. The increasing availability of structured data in Resource Description Framework (RDF) format poses new challenges and opportunities for data mining. Existing approaches to mining RDF have only focused on one specific data representation, one specific machine learning algorithm or one specific task. Kernels, however, promise a more flexible approach by providing a powerful framework for decoupling the data representation from the learning task. This paper focuses on how the well established family of kernel-based machine learning algorithms can be readily applied to instances represented as RDF graphs. We first review the problems that arise when conventional graph kernels are used for RDF graphs. We then introduce two versatile families of graph kernels specifically suited for RDF, based on intersection graphs and intersection trees. The flexibility of the approach is demonstrated on two common relational learning tasks: entity classification and link prediction. The results show that our novel RDF graph kernels used with Support Vector Machines (SVMs) achieve competitive predictive performance when compared to specialized techniques for both tasks.

1 Introduction

The RDF-formated data on the World Wide Web leads to new types of distributed information systems and poses new challenges and opportunities to data mining research. As an official standard of the World Wide Web Consortium (W3C), the *Resource Description Framework (RDF)* establishes a universal graph-based data model not intuitively suited for standard machine learning (ML) algorithms. Recent efforts of research, industry and public institutions in the context of Linked Open Data (LOD) initiatives have led to considerable amounts of RDF data sets being made available and linked to each other on the web [1]. Besides that, there is progress in research on extracting RDF from text documents [2]. Consequently, the question of systematically exploiting the knowledge therein by data mining approaches becomes highly relevant.

This paper focuses on making instances represented by means of RDF graph structures available as input to existing ML algorithms, which can solve data mining tasks relevant to RDF, such as class-membership prediction, property value prediction, link prediction or clustering. As an example, consider the mining of the social network emerging from the linked user profiles available in FOAF, a popular RDF-based vocabulary [3]. Relevant tasks in this setting include the identification of similar users (e.g. for identity resolution) or the prediction of user interests (e.g. for recommendations).

Existing approaches to mining the Semantic Web (SW) have either focused on one specific semantic data representation [4, 5] based on RDF or on one of the specific tasks mentioned above, i.e. the data representation and the learning algorithm have been devised specifically for the problem at hand [6, 7, 8, 9, 10].

In contrast, *kernels* [11] promise a highly flexible approach by providing a powerful framework for decoupling the data representation from the learning task: Specific *kernel functions* can be deployed depending on the format of the input data and combined in a "plug-and-play"-style with readily available *kernel machines* for all standard learning tasks. In this context, the challenge of learning from RDF data can be reformulated as designing adequate kernel functions for this representation. In fact, various kernel functions for general graph structures have been proposed over the last years. However, the properties of RDF graphs are different from general graphs: e.g. graphs representing chemical compounds usually have few node labels which occur frequently in the graph and nodes in these graphs have a low degree. In contrast, RDF node labels are used as identifiers occurring only once per graph and nodes may have a high degree.

In this paper, we bridge the gap between the too general and too specialized approaches to mining RDF data. We first review existing approaches for mining from semantically annotated data on the one hand and from general graphs on the other. We discuss the problems of these approaches with respect to their flexibility (e.g. applicability in various tasks, combination with different learning algorithms, different data representations etc.) and their suitability as data representations for RDF-type data. Improving on that, we introduce two versatile families of graph kernels based on intersection graphs and intersection trees. We discuss why our approach can be more easily applied by non-experts to solve a wider range of data mining tasks using a wider range of ML techniques on a wider range of RDF data sets compared to the more specialized approaches found in the related work. To the best of our knowledge, these are the first kernels which can interface between any RDF graph and any kernel machine, while exploiting the specifics of RDF. By applying standard Support Vector Machines (SVM) to RDF and RDF(S) data sets we show how our kernels can be used for solving two different learning problems on RDF graphs: property value prediction and link prediction. Despite its broader applicability we achieve comparable performance to methods which are either more specialized on a specific data format and data mining task or too general to exploit the specifics of RDF graphs efficiently.

In Section 2, we introduce foundations on kernel methods and discuss related work. In Section 3 we describe a new family of kernel functions and their applicability to learning from RDF data. In Section 4, we report experiments which demonstrate their flexibility and performance. We conclude with a discussion and outlook in Section 5.

2 Preliminaries and Related Work

In this section, we briefly introduce the required formalisms for kernel methods and present related work with respect to mining Semantic Web (SW) data. After introducing kernel methods and the learning problems arising from SW data, we discuss the applicability of general graph kernels to semantic data representations, before detailing on the existing methods for specific learning problems in the context of the SW.

2.1 Preliminaries: Kernels

The core idea behind the use of kernel functions is the decoupling of the employed learning algorithms from the representations of the data instances under investigation. Kernel-based machine learning algorithms abandon the explicit vector representations of data items by means of the *kernel function*, a similarity function which maintains a geometric interpretation as the inner product of two vectors in some, potentially even unknown, feature space. While Support Vector Machines (SVMs) for classification can safely be regarded as the best known kernel machine, kernel machines have been devised for many more supervised and unsupervised learning problems: kernel-k-means, SVM regression, one-class SVM, etc. The algorithms are formalized such that it is sufficient to access the evaluations of the inner product $\langle x, x' \rangle$ of two vectors x, x'. As a consequence, it is possible to replace the inner products $\langle \cdot, \cdot \rangle$ in the unkernelized algorithms by any valid *kernel function* which yields the same result as the inner product without the explicit representation of the training instances as vectors.

Definition 1 (Kernel Function). *Any function $\kappa : X \times X \to \mathbb{R}$ on objects x, x' from some input domain X that satisfies $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$, is a valid kernel, whereby ϕ is a mapping function (feature representation) from $\mathcal{X}$ to a feature space $\mathcal{H}$.*

Technically, the set of valid kernel functions exactly corresponds to the set of so-called *positive semi-definite functions* [11]. It can be shown that kernel functions are closed under sum, product, multiplication by a positive scalar and combination with well-known kernel modifiers. The interested reader is pointed to [11] for a more comprehensive introduction to kernel methods.

2.2 Preliminaries: Data Mining Tasks for RDF

The goal in this work is to make kernel machines available to learning tasks which use SW data as input. On RDF instances, variants of classical ML tasks, like classification, can be performed. This could be the assignment of individuals to classes, like `person`-instances to `foaf:person`-classes if this information is only partially known. The prediction of features of instances, like `foaf:topic_interest` of a person, is another classification task which we will call *property value prediction*. An important task that has been of increasing interest is *relation* or *link prediction*. In our example this could be `foaf:knows` recommendations to persons. Another important conventional data mining task applicable to RDF data is the *clustering* of instances like similar persons (sometimes also called *group detection* [12]).

Two kinds of existing approaches to mining RDF data exist: either, general graph kernels are used (see Section 2.3) or very specific approaches are devised (see Section 2.4). We argue that the first category does not exploit the properties of RDF data adequately and that approaches of the second category are, from a practical perspective, too specific with respect to the learning problem, data representation, or dataset.

2.3 Related Work: General Graph Kernels

Graphs present a very generic model for representing various kinds of data. Therefore, the problem of making graphs amenable for kernel machines has received considerable

interest. In the spirit of the *Convolution Kernel* by Haussler [13], which represents a generic way of defining kernels, kernel functions for general graphs have been devised by counting common subgraphs of two graphs. While the subgraph isomorphism problem, i.e. the problem of identifying whether a given input graph contains another input graph as its subgraph, is known to be NP-complete, the search for common subgraphs with specific properties can often be performed more efficiently. Thus, kernel functions have been defined for various substructures such as walks [14], cycles [15] and trees [16]. Any of these kernels is applicable to RDF data as long as it is defined on directed, labeled graphs. While these kernel functions are applicable in a wide range of applications, they do not exploit the specific properties of RDF data. RDF graphs are directed, labeled graphs in which nodes may be identified by their label, their URI.

2.4 Related Work: Specialized Methods for Mining Semantic Data

Besides the general graph kernels, specific approaches for mining from Semantic Data have been developed. Huang et al. [8] have proposed a link prediction method which is based on the matrix completion of the relation matrix. Thor et al. [10] have proposed a link prediction method which is based on finding frequent patterns in the data graph. Rettinger et al. [7] have proposed a method for integrating ontology-based constraints with statistical relational learning. These approaches are tailored towards specific applications and thus do not expose the general applicability of kernel methods. Fanizzi and d'Amato [4] propose a declarative kernel for semantic concept descriptions in the description logic (DL) $\mathcal{ALC}$. Structurally, these kernels are based on a representation of $\mathcal{ALC}$ concepts in normal form. Fanizzi et al. [5] have proposed a different set of kernels, which can be applied directly to individuals. These kernel functions are based on the analysis of the membership of the individuals in a set of classes. These kernel functions are tailored towards specific data representations and rely on clean, DL-based formal ontologies which, in contrast to lightweight RDF-based data, only constitute a very small part of the "semantic" data sources published. Bicer et al. [9] have proposed a very general family of kernel functions which is based on a set of logical clauses that are generated and selected automatically. However, this approach can not be integrated "plug-and-play"-style with existing kernel machines as these have to be adapted in order to select a relevant subset of the defined features. Finally, Bloehdorn and Sure [6] have proposed a set of kernel functions for individuals based on a manual feature definition. This means that relevant kernels and corresponding features for the learning task and the dataset at hand have to be defined upfront.

The goal of our work is to define kernel methods for RDF which can be used with any existing kernel machine, which is not restricted to specific kinds of data, which can readily be applied to any RDF dataset, but which still exploits the specific properties of RDF-type data (in contrast to general graph data).

3 Kernel Functions for RDF Graphs – Design and Implementation

In this section, we present our core contribution: two classes of kernel functions based on *intersection graphs* and *intersection trees*, specifically tailored to the properties of RDF. We thereby rely on feature sets based on the graph structure underlying RDF data.

Definition 2 (RDF Graph). *An RDF graph is defined as a set of triples of the form $G = \{(s, p, o)\} = (V, E)$, where the subject $s \in V$ is an entity, the predicate p denotes a property, and the object $o \in V$ is either another entity or, in case of a relation whose values are data-typed, a literal. The vertices $v \in V$ of G are defined by all elements that occur either as subject or object of a triple. Each edge $e \in E$ in the graph is defined by a triple (s, p, o): the edge that connects s to o and has label p.*[3]

Generally, we will look at RDF entities as the instances for learning. For example, two sets of entities, identified by their Uniform Resource Identifiers (URIs) could be positive and negative classes in a classification scenario. The argument entities' neighborhood in the overall RDF graph forms the basis for their kernel-induced feature representations. Essentially, all proposed kernel functions are thus based on a neighborhood graph which is obtained by a breadth-first search *up to depth k* starting from the entity of interest. We have defined two versions of the neighborhood graph: the *intersection graph* (see Section 3.1) and the *intersection tree* (see Section 3.2).

We define RDF kernels in a similar manner to other graph kernels by adopting the idea of counting subgraphs with a specific structure in the input graphs. The essential difference is that, as RDF builds on unique node labels, each RDF subgraph can occur at most once in the input graph. This is not the case in general graphs, where it is common that several nodes carry the same label – thus yielding potentially several equivalent subgraphs. Therefore, when calculating the kernel function between two RDF graphs, it is not necessary to identify the interesting structures and their frequencies in the two graphs separately. Instead, it is sufficient to analyze a single structure which contains the features of interest *common* in both input graphs.[4] For each of the definitions of the neighborhood graphs sketched above, we have defined a way of representing their *common* structures, which are used as basis for the two families of kernel functions we define: In Section 3.1 we will present kernel functions which are based on *intersection graphs* (obtained from two instance graphs), in Section 3.2 kernel functions based on *intersection trees* (on the basis of instance trees) will be presented.

3.1 Kernel Functions Based on Intersection Graphs

The intersection graph of two graphs is a graph containing all the elements the two graphs have in common.

Definition 3 (Intersection Graph). *The intersection graph $G_1 \cap G_2$ of two graphs G_1 and G_2 is defined as: $V(G_1 \cap G_2) = V_1 \cap V_2$ and $E(G_1 \cap G_2) = \{(v_1, p, v_2) | (v_1, p, v_2) \in E_1 \wedge (v_1, p, v_2) \in E_2\}$.*

Note that if the intersection graph contains a given subgraph, this subgraph is also a subgraph of each of the two input graphs. Inversely, if a subgraph is contained in both

[3] Note that this defines a multigraph which allows for multiple edges between the same two nodes, as long as the type of the relation is different.

[4] Gärtner et al. [14] have proposed kernel functions which are based on counting common structures in the *direct product graph*. In the case of graphs with unique node labels, like RDF, this is equivalent to what we call the *Intersection Graph* which we define in Section 3.1.

instance graphs, it is part of the intersection graph. Thus, calculating a kernel function based on a set of subgraphs can be reduced to constructing the intersection graph in the first step and then counting the substructures of interest therein. We have defined kernels based on implicit feature sets based on walks, paths, and (connected) subgraphs:

Edge-Induced Subgraphs. The set of edge-induced subgraphs qualifies as a candidate feature set.

Definition 4 (Edge-Induced Subgraph). *An edge-induced subgraph of $G = (V, E)$ is defined as $G' = (V', E')$ with $E' \subseteq E$ and $V' = \{v \mid \exists u, p : (u, p, v) \in E' \vee (v, p, u) \in E'\}$. We denote the edge-induced subgraph relation by $G' \subseteq G$.*

The set of edge-induced subgraphs is a valid feature set, as counting edge-induced subgraphs in the intersection graph is equivalent to performing the feature mapping explicitly and calculating the dot product of the two feature vectors.

Walks and Paths. Connected elements within the intersection graphs are likely to yield more interesting results than a set of arbitrary relations taken from the intersection graph. We have therefore defined additional kernels whose features are restricted to subsets of all edge-induced subgraphs. We have focused on *walks* and *paths* as interesting subsets, as they represent property chains in RDF.

Definition 5 (Walk, Path). *A* walk *in a graph $G = (V, E)$ is defined as a sequence of vertices and edges $v_1, e_1, v_2, e_2, \ldots, v_n, e_n, v_{n+1}$ with $e_i = (v_i, p_i, v_{i+1}) \in E$. The length of a walk denotes the number of edges it contains.*

A path *is a walk which does not contain any cycles i.e. a walk for which the additional condition $v_i \neq v_j \forall i \neq j$ holds. We denote the set of walks of length l in a graph G by $walks_l(G)$, the paths up to length l by $paths_l(G)$.*

Definition 6 (Walk Kernel, Path Kernel). *The Walk Kernel for maximum path length l and discount factor $\lambda > 0$ is defined by $\kappa_{l,\lambda}(G_1, G_2) = \sum_{i=1}^{l} \lambda^i |\{w | w \in walks_i(G_1 \cap G_2)\}|$. Analog the Path Kernel $\kappa_{l,\lambda}(G_1, G_2) = \sum_{i=1}^{l} \lambda^i |\{p | p \in paths_i(G_1 \cap G_2)\}|$.*

The corresponding feature space consists of one feature per walk (resp. path). In the definition, the parameter $\lambda > 0$ serves as a discount factor and allows to weight longer walks (paths) different from shorter ones. If $\lambda > 1$ then longer walks (paths) receive more weight, in case of $\lambda < 0$ shorter ones contribute more weight.

As paths and walks are edge-induced substructures of a graph, the validity of the proposed kernel functions can be shown using the same argument as for edge-induced subgraphs. The kernel function can be calculated iteratively by constructing walks of length i as extension of walks of length $i - 1$. In each iteration an edge is appended at the end of the walks found in the previous iteration. For counting paths, the condition that $v_i \notin \{v_1, \ldots, v_{i-1}\}$ has to be added when constructing the paths of length i.

A different approach for calculating these kernel functions is based on the powers of the intersection graph's adjacency matrix. The adjacency matrix M is a representation of a graph in the form of a matrix with one row and one column per node in the graph and entries $x_{ij} = 1$ if the graph contains an edge from node i to node j, 0 otherwise.

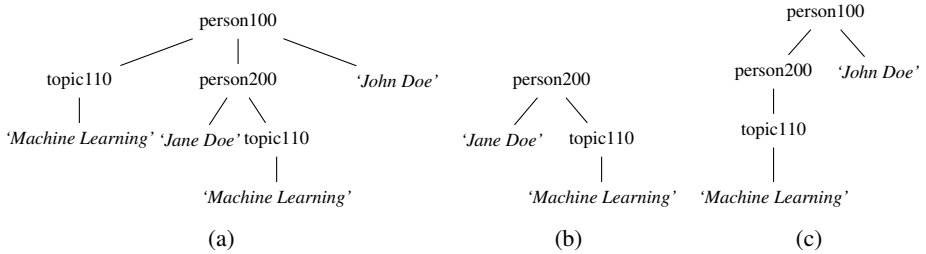

Fig. 1: (a) Example of an instance tree, (b) example of a complete subtree and (c) example of a partial subtree of the instance tree.

Each entry x_{ij} of the k^{th} power of M can be interpreted as the number of walks of length k existing from node i to node j. Therefore, the number of walks up to length k in the graph can be obtained as $\sum_{i=1}^{k}\sum_{j=1}^{n}\sum_{l=1}^{n}(M^i)_{jl}$. By setting the elements x_{jj} of M^i to 0, this formula can also be used for the path kernel.[5]

3.2 Kernel Functions Based on Intersection Trees

The kernel functions presented in the previous section are based on the calculation of the intersection graph. The use of the intersection graph may become problematic as its calculation is potentially expensive: the whole instance graph for each entity has to be extracted and the two graphs have to be intersected explicitely. However, the size of the instance graph grows exponentially with the number of hops crawled from the entity to build up the instance graph. Merging the two steps of extracting the instance graphs and intersecting them is not directly feasible: Consider an entity e which can be reached within k hops from both entities e_1 and e_2, but through different paths. In this case, e would be part of the intersection graph, but it would not be reachable from either e_1 or e_2 in this graph. In the following, we present a different way of extracting common neighborhoods of two entities, which enables a direct construction of the common properties, without building the instance graphs. This alternative method is based on the use of instance trees instead of instance graphs. Instance trees are obtained based on the graph expansion with respect to an entity of interest e (as for example defined in [17]).

Definition 7 (Graph Expansion). *The expansion $X(e)$ of a graph G with respect to entity e is a tree defined as follows: (i) If e does not have any successors, then $X(e)$ is the tree consisting only of the node e. (ii) If $v_1, \ldots, v_n$ are the successors of e, then $X(e)$ is the tree $(e, X(v_1), \ldots, X(v_n))$ (in prefix notation).*

[5] Gärtner et al. [14] use this approach for counting walks up to infinite length by calculating the limes $k \to \infty$ of the matrix power series. They also use a weight factor to give different weight to walks of different length. However, for the matrix power series to converge, their weight factor has to be smaller than 1. Thus, in their kernel it is only possible to give smaller weight to larger structures. In the case of RDF, it may however be preferable to give more weight to larger structures as those convey more meaning.

The graph expansion can grow infinitely if the graph contains cycles. To avoid this problem and to limit the size of the obtained trees the graph expansion is bound by depth d. While in the original RDF graph, node labels were used as identifiers, i.e. each node label occurred exactly once, this is not true in the expanded graph. If there is more than one path from entity e to another entity e', then e' will occur more than once, and thus the label of e' is not unique anymore. An intersection of the instance trees leads to an intersection tree in the same spirit as the intersection graph. We introduce two changes to the instance trees as obtained by direct expansion from the data graph:

Definition 8 (Intersection Tree). *An intersection tree $IT(G, e_1, e_2, d)$ is the tree obtained through the following steps:*

1. *Intersect the instance trees of depth d for entities e_1 and e_2.*
2. *Replace all occurrences of labels of e_1 and e_2 by a new dummy label not occuring elsewhere in the graph.*
3. *Retain only those parts of the intersection which are connected to the root element.*

The intersection tree can be extracted directly from the data graph without constructing the instance trees explicitely. Therefore, the intersection tree can be directly obtained using Algorithm 23. The algorithm directly extracts the intersection tree $it_d(e_1, e_2)$ from the data graph. Starting from the two entities e_1 and e_2 the intersection tree is built using breadth-first search. Two cases have to be distinguished. In cases where one of the entities e_1 or e_2 is found a new node is added to the tree with a dummy label. For nodes with this label, the common relations of e_1 and e_2 are added as children. The second case are all nodes which do not correspond to one of the entities: for them, a new node with the node's URI respective label is added to the tree, the children of these latter nodes are all relations of this node in the data graph.

As in the case of the intersection graphs, our proposed kernel functions are based on counting elements in the intersection trees. The features of interest are restricted to features which contain the root of the tree. This is because features which are not connected to the root element may be present in each instance tree, but may by construction not be part of the intersection tree.

Full Subtrees. The first kernel function we propose based on the intersection tree is the *full subtree kernel*, which counts the number of full subtrees of the intersection tree $it_d(e_1, e_2)$, i.e. of the intersection tree of depth d for the two entities e_1 and e_2. A full subtree of a tree t rooted at a node v is the tree with root v and all descendants of v in t (see Figure 1 for an example).

Definition 9 (Full Subtree Kernel). *The* Full Subtree kernel *is defined as the number of full subtrees in the intersection tree. Subtrees of different height are weighted differently using a discount factor λ: $\kappa_s t(e_1, e_2) = st(root(it_d(e_1, e_2)))$ where $st(v) = 1 + \lambda \sum_{c \in children(v)} st(c)$.*

The corresponding feature mapping consists of one feature per subtree. Counting the number of full subtrees in the kernel is equivalent to counting the walks starting at the root of the intersection tree. The full subtree kernel is valid due to this equivalence.

Algorithm 1: Extraction of an intersection tree of depth d for entities e_1 and e_2

Input: entities e_1, e_2
maximum tree depth d
Data: RDF data graph $G = (V, E)$, where labels of entities e_1 and e_2 are replaced by source
Result: $tree$: Graph expansion $X(e_1 \cap e_2)$ of depth d

$tree \leftarrow$ new Node("source",0)
$newLeaves \leftarrow \{tree\}$
for $i = 1 : d - 1$ **do**
  $leaves \leftarrow newLeaves$;
  for $leaf \in leaves$ **do**
    if $leaf$*.label="source"* **then**
      $ce \leftarrow \{e_i | (e_1, p, e_i) \in G \wedge (e_2, p, e_i) \in G\}$
      for $p : (e_1, p, e_2) \in G \wedge (e_2, p, e_1) \in G$ **do**
        ce.add(new Node("source"))
    else
      $ce \leftarrow \{e_i | (leaf, p, e_i) \in G\}$
    for $c \in ce$ **do**
      if $c \in \{e_1, e_2\}$ **then**
        $label$="source"
      else
        $label$=c.uri
      $child \leftarrow$ new Node($label$, $leaf.depth + 1$)
      $leaf$.addChild($child$)
      $newLeaves$.add($child$)

Partial Subtrees. Given a tree $T = (V, E)$, its partial subtrees are defined by subsets $V' \subset V$ and $E' \subset E$ such that $T' = (V', E')$ is a tree. We propose to define a kernel function which counts the number of partial subtrees in the intersection tree $it_d(e_1, e_2)$ who are rooted at the root of $it_d(e_1, e_2)$.

Definition 10 (Partial Subtree Kernel). *The* Partial Subtree Kernel *is defined as the number of partial trees that the intersection tree contains. A discount factor λ gives more or less weight to trees with greater depth: $\kappa_{pt}(e_1, e_2) = t(root(it_d(e_1, e_2)))$ where t is defined as: $t(v) = \prod_{c \in children(v)} (\lambda t(c) + 1)$. The function $t(v)$ returns the number of partial subtrees with root v that the tree rooted at v contains weighted by depth with a discount factor λ.*

The corresponding feature space consists of one feature per partial tree up to depth d with root e_i replaced by a dummy node in the data graph. The value of each feature is the number of times a partial tree occurs.

4 Evaluation

To show the flexibility of the proposed kernels, we have conducted evaluations on real world data sets in two learning tasks: a property value prediction task and a link prediction task. The kernel functions are implemented in Java using JENA[6] for processing RDF. For the Property Value Prediction, we used SVM*light* [18] with the JNI Kernel Extension.[7] The Link Prediction problem we used the ν-SVM implementation of LIBSVM [19].

4.1 Property Value Prediction

In a first evaluation setting, we applied our kernels to property value prediction problems. The task consists in predicting which of the possible values a property should have for a specific entity. The task thus consists in classifying the entities into the class corresponding to the predicted property value. Our approach is compared to two kernels that have been deviced for general graphs: the Weisfeiler-Lehman kernel [16] and the Gaertner-kernel [14]. The Weisfeiler-Lehman kernel has been chosen as a state-of-the-art graph kernel which can be computed very efficiently. It is based on counting common subtree-patterns in the input graphs. The Gaertner kernel is closely related to the kernel function we present here: it is based on the analysis of the direct product graph which is equivalent to the intersection graph in the case of RDF data. Additionally, the kernel functions presented by Bloehdorn and Sure [6] are used for comparison: these are kernel functions manually defined for a specific dataset.

Data sets. The first evaluation set uses data from the SWRC ontology [20] and the metadata available in the Semantic Portal of the institute AIFB. The ontology models key concepts within a research community. The evaluation data consists of 2,547 entities of which 1,058 belong to the person class. 178 of these persons are affiliated with one of the research groups at AIFB. There are 1232 instances of type publication, 146 of type research topic and 146 of type project. The entities are connected by a total of 15,883 relations. Additionally, there are 8,705 datatype properties, i.e. properties linking an entity to a literal. The evaluation setting defined in [6] consists in classifying staff members with respect to their affiliation to research groups. All relations denoting the affiliation of a person with a research group were deleted from the training data.

In a second setting, a larger dataset consisting of Friend of a Friend (FOAF) descriptions of people from the community website LiveJournal.com was used for experiments. The dataset describes 22745 people and their social networks, as well as some personal information (location, school they attended, online accounts, their birthdays and the number of posts on their blog). Note that birth dates and number of blog posts are reduced to a small number of discrete states. For example, the precise age was replaced by one of four newly introduced age classes. For the evaluation, we removed all relations of type $dateOfBirth$ from the dataset and tried to learn this relation afterwards, i.e. the goal was to predict for all 1567 people with available age information to which of the 4 age classes they should belong.

[6] `http://jena.sourceforge.net`

[7] `http://people.aifb.kit.edu/sbl/software/jnikernel/`

Table 1: Evaluation results for Property Value Prediction

Kernel configuration				SWRC results		LiveJournal results	
Kernel	Instance Depth	Max. Size	Discount	Error	F1 measure	Error	F1 measure
Bloehdorn/Sure [6]				0.0449	0.7237	–	–
Weisfeiler-Lehman [16]	2	2		0.1096	0.0000	0.2215	0.4084
Gaertner [14]	2		0.5	0.0590	0.6625	0.3824	0.5344
Paths/Walks	2	1	1	0.0576	0.6318	0.2125	0.4096
Paths	2	2	1	0.0576	0.6318	0.1946	0.4192
Walks	2	2	1	0.0576	0.6318	0.1948	0.4175
Full Subtree	2		1	0.0548	0.6543	0.1902	0.4018
Partial Subtree	2		0.01	0.1025	0.3247	0.1866	0.3286
Edges (no schema)	2			0.0478	0.7488	–	–
Walks (no schema)	2	2	1	0.0478	0.7488	–	–
Paths (no schema)	2	2	1	0.0478	0.7488	–	–
Full Subtree	2		1	0.0548	0.6687	–	–

Compared approaches. All results reported in Table 1 were obtained using leave-one-out cross-validation. The trade-off parameter c of the Support Vector Machine learning was set to 1 in all experiments. We compare the results obtained using the kernel functions defined in Sect. 3 to the Weisfeiler-Lehman kernel and the Gärtner kernel. We applied these graph kernels to the instance graphs of depth 2 which were extracted from the RDF data graph. We chose the parameters of the compared approaches according to the best setting reported in the literature: The maximum depth of trees in the Weisfeiler-Lehman kernel was set to 2, the discount factor for longer walks in the Gärtner kernel was set to 0.5. On the SWRC dataset, the best configuration obtained by Bloehdorn and Sure [6] in the original paper was used for comparison. This kernel configuration, denoted by *sim-ctpp-pc* combines the common class similarity kernel described in their paper with object property kernels for the *workedOnBy*, *worksAtProject* and *publication*. Additionally, also on the SWRC dataset, we compared the performance of our kernels on the whole data set (including the schema) to a setting where all relations which are part of the schema were removed (lowest part in Table 1).

Discussion. Our results show that with specific parameters our kernels reach comparable error and higher F1-measure than the kernels proposed by Bloehdorn and Sure. Considering that our approach is generic and can be used off the shelf in many scenarios, while their kernel function was designed manually for the specific application scenario, this is a positive result. Our kernel functions also perform well with respect to other graph kernels: The Weisfeiler-Lehman kernel is not able to separate the training data in the SWRC dataset as it can match only very small structures. While the Gaertner kernel achieves results which are comparable to our results, its calculation is more expensive - due to the cubic time complexity of the matrix inversion. Our results show that the walk kernel and the path kernel perform better in terms of classification errors when the maximum size of the substructures are increased. As for the partial subtree kernel it turned out that parameter tuning is difficult and that the results strongly de-

pend on the choice of the discount factor. Last but not least, a surprising result is that the kernels which are based on the intersection graph perform better on the reduced data set which does not contain the schema information. Our explanation for this is that the intersection graph contains part of the schema and thus produces a similar overlap for many of the instances. Further results which we do not report here show that with increasing maximum size k of the considered substructures precision is increased and recall is decreased. This effect may be due to the effect that bigger structures describe instances in a more precise way, thus refining the description of elements belonging to the class but losing generality of the class description on the other hand.

4.2 Link Prediction

In a second scenario we evaluated our kernel functions on link prediction problems. We formalized the link prediction as a binary classification problem where the goal is to predict whether a link exists between two entities. The classifier is learned for a specific relation. In this problem, we require a kernel function for the comparison of relation instances. We propose a link kernel which is based on the comparison of entities:

$$\kappa((s_1, o_1), (s_2, o_2)) = \alpha\kappa_s(s_1, s_2) + \beta\kappa_o(o_1, o_2)$$

This is a valid kernel function for $\alpha, \beta \geq 0$ as the space of valid kernel function is closed under sum and multiplication with a positive scalar [11]. Any kernel functions for RDF entities may be used as subject and object kernel (κ_s and κ_o).

Datasets. We evaluated the link prediction approach on two datasets. The first evaluation setting uses the SWRC dataset as for property value description. In this scenario the goal is to predict the `affiliation` relation, i.e. to decide for a tuple of person and research group whether the given person works in the given research group. The second dataset is a reduced version of the LiveJournal dataset used for age prediction. The reduced dataset contains descriptions of 638 people and their social network. Overall, there are 8069 instances of the `foaf:knows` relation that is learned in this setting.

Evaluation method. RDF is based on the open-world assumption, which means that if a fact is not stated, it may not be assumed to not hold. Rather, the truth value of this relation is unknown. This is problematic as no negative training and test data is available. As to the training phase, some initial experiments using one-class SVM [21] did not yield promising results. We therefore considered some negative training data to be necessary for the evaluation. Therefore, the positive training instances were complemented with an equal number of unknown instances. We consider this number of negative instances a good trade-off between the number of assumptions made and the gain in quality of the trained classifier. The obtained training set is a balanced dataset which does not present any bias towards one of the classes.

Evaluation may not be based on positive or negative classifications due to the absence of negative data. Thus, a ranking-based approach to the evaluation was chosen: positive instances should be ranked higher in the result set than negative ones. For evaluation, a modified version of 5-fold cross validation was used: the positive instances were split in five folds, each fold was complemented with an equal number of unknown instances and the unused unknown instances were used as additional test data. Each

Table 2: Evaluation Results for Link Prediction

Kernel configuration				Results for SWRC		Results for LiveJournal	
Kernel	Inst. depth	α/β	Param. SUNS	NDCG	bpref	NDCG	bpref
SUNS-SVD			20	0.8022	0.3844	0.7871	0.2889
SUNS-Regularized			1	0.4038	0.0538	0.5446	0.0125
Weisfeiler-Lehman	2	1		0.9443	0.8303	0.9991	0.9963
Common Subtrees	2	0.5		0.9316	0.8363	0.9724	0.9056
Common Subtrees	2	1		0.9340	0.8438	0.9886	0.9680
Common Subtrees	2	2		0.9548	0.8538	0.9694	0.8948
Common Subtrees	2	3		0.9271	0.7831	0.9741	0.9006
Common Subtrees	2	5		0.8387	0.6313	0.9744	0.8945

model was trained on four folds and evaluated on the fifth fold and the additional test data. NDCG [22] and bpref [23] were used as evaluation measures.

Compared approaches. We have compared our approach to the SUNS approach presented by Huang et al. [8] and to the link kernel with the Weisfeiler-Lehman kernel [16] as entity kernel. This approach is based on a singular value decomposition of the relation matrix and generates predictions based on the completed matrix. We experimented with different settings for the parameter of SUNS, where the range of the parameter was taken from Huang et al. [8]. However, our evaluation procedure is substantially different to theirs. We report the best results which we obtained. Results are reported in Table 2. The settings of the Weisfeiler-Lehman kernel are the same as in the property value prediction task. For all SVM-based evaluations presented here, the parameter ν was set to 0.5.

Discussion. Our results show that our Link Prediction approach can outperform the SUNS approach in terms of NDCG and bpref. Partly, this may be due to the chosen evaluation procedure: the SUNS approach can only deal with those relation instances which were present in the training phase: as only 80% of the available data were used for training, some instances of the domain or range may not occur in the training data and thus no prediction is obtained for those. To obtain a complete ranking, a minimum value was assigned to these instances[8] Additionally, a pruning step is part of the preprocessing steps of SUNS, which removes elements with few connections. In comparison with the Weisfeiler-Lehman kernel our kernel functions achieve comparable results on the SWRC dataset. As to the relation of α and β in the link kernel, best results are obtained for $\frac{\alpha}{\beta} = 2$ in the SWRC dataset and for $\frac{\alpha}{\beta} = 1$ in the LiveJournal dataset. We suppose that the higher importance of the subject in the SWRC dataset is due to the smaller number of objects in the range of the relation. In contrast, in the LiveJournal dataset there is an equal number of elements in the domain and the range of the relation. Another finding from additional experiments which we do not report here is that the quality of the model increases with growing ν. This means that a complexer model achieves better generalisation resuts on both datasets.

[8] This explains the very low bpref achieved by the SUNS approach: all positive instances for which no prediction could be obtained are ranked lower than all negative instances.

5 Conclusion

With the advent of the Resource Description Framework (RDF) and its broad uptake, e.g. within the Linked Open Data (LOD) initiative, the problem of mining from semantic data has become an important issue. In this paper, we have introduced a principle approach for exploiting RDF graph structures within established machine learning algorithms by designing suitable kernel functions which exploit the specific properties of RDF data while remaining general enough to be applicable to a wide range of learning algorithms and tasks. We have introduced two versatile families of kernel functions for RDF entities based on intersection graphs and intersection trees and have shown that they have an intuitive, powerful interpretation while remaining computationally efficient. In an empirical evaluation, we demonstrated the flexibility of this approach and show that kernel functions within this family can compete with hand-crafted kernel functions and computationally more demanding approaches.

In the future, we plan to extend this framework to other substructures that can be shown to be present in intersection graphs resp. intersection trees and to apply it to new tasks and other LOD data sets.

Acknowledgments. The work was funded by the EU project XLike under FP7.

References

[1] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[2] I. Augenstein, S. Padó, and S. Rudolph. Lodifier: Generating linked data from unstructured text. In *Proc. of the 9th Extended Semantic Web Conference (ESWC'12)*, Lecture Notes in Computer Science. Springer, 2012.

[3] D. Brickley and L. Miller. FOAF vocabulary specification. Technical report, FOAF project, 2007. Published online on May 24th, 2007 at `http://xmlns.com/foaf/spec/20070524.html`.

[4] N. Fanizzi and C. d'Amato. A declarative kernel for ALC concept descriptions. In *Foundations of Intelligent Systems, 16th International Symposium,*, pages 322–331, 2006.

[5] N. Fanizzi, C. d'Amato, and F. Esposito. Statistical learning for inductive query answering on OWL ontologies. In *Proc. of the 7th International Semantic Web Conference, ISWC2008*, pages 195–212, 2008.

[6] S. Bloehdorn and Y. Sure. Kernel methods for mining instance data in ontologies. In *Proc. of the 6th Int. Semantic Web Conference and the 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007)*, pages 58–71, 2007.

[7] A. Rettinger, M. Nickles, and V. Tresp. Statistical relational learning with formal ontologies. In *Proc. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2009)*, volume 5782 of *Lecture Notes in Computer Science*, pages 286–301. Springer, Berlin–Heidelberg, Germany, 2009. ISBN 978-3-642-04173-0.

[8] Y. Huang, V. Tresp, M. Bundschus, A. Rettinger, and H.-P. Kriegel. Multivariate structured prediction for learning on semantic web. *Proc. of the 20th International Conference on Inductive Logic Programming (ILP 2010)*, 2010.

[9] V. Bicer, T. Tran, and A. Gossen. Relational kernel machines for learning from graph-structured rdf data. In *Proc. of the 8th Extended Semantic Web Conference (ESWC'11)*, volume 6643 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2011.

[10] A. Thor, P. Anderson, L. Raschid, S. Navlakha, B. Saha, S. Khuller, and X.-N. Zhang. Link prediction for annotation graphs using graph summarization. In *Proc. of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, pages 714–729. Springer-Verlag, 2011. ISBN 978-3-642-25072-9.

[11] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[12] L. Getoor, N. Friedman, D. Koller, A. Pferrer, and B. Taskar. Probabilistic relational models. In *Introduction to Statistical Relational Learning*. MIT Press, 2007.

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

[14] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proc. of the 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop (COLT/Kernel 2003)*, pages 129–143, 2003.

[15] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2004)*, pages 158–167. ACM Press, New York, NY, USA, 2004.

[16] N. Shervashidze and K. Borgwardt. Fast subtree kernels on graphs. In *Advances in Neural Information Processing Systems 22*. 2009.

[17] R. H. Güting. *Datenstrukturen und Algorithmen*. B.G. Teubner Stuttgart, 1992.

[18] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*, 1999.

[19] C.C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[20] Y. Sure, S. Bloehdorn, P. Haase, J. Hartmann, and D. Oberle. The SWRC ontology - Semantic Web for research communities. In *Proc. of the 12th Portuguese Conference on Artificial Intelligence (EPIA 2005)*, pages 218–231, 2005.

[21] B. Schölkopf, J.C. Platt, J. Shawe-Taylor, A. Smola, and R.C. Williamson. Estimating the support of a high-dimensional distribution. Technical report, 1999.

[22] K. Järvelin and J. Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *Proc. of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 41–48. ACM, 2000. ISBN 1-58113-226-3.

[23] C. Buckley and E.M. Voorhees. Retrieval evaluation with incomplete information. In *Proc. of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'04)*, pages 25–32. ACM, 2004.