A Fast Approximation of the Weisfeiler-Lehman Graph Kernel for RDF Data

Gerben K.D. de Vries

System and Network Engineering Group, Informatics Institute,
University of Amsterdam, The Netherlands
g.k.d.devries@uva.nl

Abstract. In this paper we introduce an approximation of the Weisfeiler-Lehman graph kernel algorithm aimed at improving the computation time of the kernel when applied to Resource Description Framework (RDF) data. Typically, applying graph kernels to RDF is done by extracting subgraphs from a large RDF graph and computing the kernel on this set of subgraphs. In contrast, our algorithm computes the Weisfeiler-Lehman kernel directly on the large RDF graph, but still retains the subgraph information. We show that this algorithm is faster than the regular Weisfeiler-Lehman kernel for RDF data and has at least the same performance. Furthermore, we show that our method has similar or better performance, and is faster, than other recently introduced graph kernels for RDF.

Keywords: Resource Description Framework (RDF), Graph Kernels, Weisfeiler-Lehman.

1 Introduction

Machine learning techniques have been widely used to populate the semantic web, i.e. to create linked data. In contrast, there has been relatively little research into learning directly from the semantic web. However, the amount of linked data available is becoming larger and larger and provides interesting opportunities for data-mining and machine learning.

Kernel methods [1,2] are popular machine learning techniques for handling structured data. To deal with data structured as graphs, graph kernels, such as described in [3] and [4], have been developed.

The representation/storage format of the semantic web is the Resource Description Framework (RDF). The RDF format essentially represents a graph. Therefore, learning from RDF can potentially be accomplished using graph kernel methods on RDF. Research on this is in its infancy and, to the best of our knowledge, there currently exists one paper [5] on this topic. In [5] the authors introduce two types of graph kernels, designed for RDF, and compare these to general graph kernels in two tasks. The authors conclude that the introduced kernels work better or just as well as the general graph kernels. For the application of most of the graph kernels, instances are represented as (small) subgraphs extracted from a larger RDF graph.

Graph kernel computation is in general slow, since it is often based on computing some form of expensive (iso)morphism between graphs. In this paper we present an approximation of the Weisfeiler-Lehman graph kernel [4] to speed up the computation of the kernel on RDF data. This approximation exploits the fact that the subgraph instances for RDF learning tasks are usually extracted from the same large RDF graph. We test this kernel on a number of learning tasks with RDF data and compare its performance to the graph kernels designed for RDF described in [5].

Kernel methods have not been widely applied to RDF data. Earlier attempts have been [6] and [7]. In [6] kernels are manually designed by selecting task relevant properties and relations for the instances, and then incorporating these in the kernel measure. Kernels are built from RDF using inductive logic programming rules in [7]. The approach to learning from RDF using graph kernels is more generally applicable than both these methods. Other attempts at learning from semantic web data are based on description logic [8,9]. These approaches are applicable to a smaller part of the semantic web, since not everything on the semantic web is nicely formalized as description logic ontologies, whereas nearly everything on the semantic web is available as RDF. Other specifically tailored approaches for data-mining form the semantic web are, for instance, using statistical relational learning [10].

The rest of this paper is structured as follows. Section 2 introduces our adaption of the Weisfeiler-Lehman graph kernel for RDF data. We present our experiments with this kernel in Sect. 3. Finally, we end with conclusions and suggestions for future work.

2 Weisfeiler-Lehman Graph Kernel for RDF

In the following section we first briefly introduce RDF data. Then we define the regular Weisfeiler-Lehman kernel for graphs. Finally we introduce our adaption of this kernel to speed up computation on RDF data.

2.1 The Resource Description Framework

The Resource Description Framework (RDF) is the foundation for knowledge representation on the semantic web. It is based on the idea of making statements about resources in a subject-predicate-object form. Such expressions are dubbed *triples*. The RDF specification[1] defines a number of classes, for the subjects and objects, and properties, for the predicates. Moreover, users can, and should add their own classes and objects.

For example, suppose that we have an ontology about fruits, called fruit. Then the RDF triple fruit:Pear-rdfs:SubClassOf-fruit:Fruit expresses the fact the the class of Pear is a sub-class of the class of Fruit. And the triple fruit:elstar-rdf:type-fruit:Apple expresses the fact that an elstar is an instance of the class Apple. The

[1] `http://www.w3.org/standards/semanticweb/`

colon notation is used to indicate from which ontology a class or property is used, i.e. the class Pear comes from the example fruit ontology that we created ourselves, but the property type is defined by the RDF standard. These notations are shorthands for full-fledged Universal Resource Identifiers (URI) that uniquely identify the specific ontology used and where to find it, thereby forming the backbone of the semantic web.

RDF resources, i.e. the uri:value type of statements, can occur as subject, predicate and object in a triple. This means that they can be both vertex and edge in a graph at the same time. Therefore, formally, RDF represents a hypergraph. However, in practice interpreting RDF as an easier to handle directed multigraph does not lead to problems for applying graph kernels.

RDF is used as a representation scheme for more expressive knowledge representation formalisms such as the Web Ontology Language (OWL) and RDF Schema (RDFS). Therefore, RDF triple-stores, such as SESAME[2], often include a reasoning engine which allows the automatic derivation of new triples, using these more expressive formalisms.

In machine learning/data-mining for RDF, arguably, the most straightforward way to represent instances is as a set of RDF triples, or an RDF graph. For example, for each fruit that is of rdf:type fruit:Apple or fruit:Pear we can collect the RDF triples that describe properties of this fruit and then these sets of triples are our instances, which we can use to train a classifier for apples and pears.

2.2 Regular Weisfeiler-Lehman Graph Kernel

The Weisfeiler-Lehman Subtree graph kernel, from now on the Weisfeiler-Lehman kernel, is a state-of-the-art, efficient kernel for graph comparison introduced in [11] and elaborated upon in [4]. The kernel computes the number of subtrees shared between two graphs by using the Weisfeiler-Lehman test of graph isomorphism. The rewriting procedure underlying the Weisfeiler-Lehman kernel is given in Algorithm 1, which is taken from [4]. The idea of the rewriting process is that for each vertex we create a multiset label based on the labels of the neighbors of that vertex. This multiset is sorted and together with the original label concatenated into a string, which is the new label. For each unique string a new (shorter) label is introduced and this replaces the original vertex label. Note that the sorting of the strings in step 3 is not necessary, but provides a simple way to create the label dictionary f. The rewriting process can be efficiently implemented using counting sort, for which details are given in [4].

Using the rewriting techniques of Algorithm 1 it is straightforward to define a kernel, given in in Definition 1.

Definition 1. *Let $G_n = (V, E, l_n)$ and $G'_n = (V', E', l_n)$ be the n-th iteration rewriting of the graphs G and G' using Algorithm 1 and h the number of iterations. Then the Weisfeiler-Lehman kernel is defined as:*

[2] `http://www.openrdf.org/`

Algorithm 1. Weisfeiler-Lehman Relabeling

```
Input graphs $G = (V, E, \ell), G' = (V', E', \ell)$ and number of iterations $h$
Output label functions $l_0$ to $l_h$
Comments $M_n(v)$ are sets of labels for a vertex $v$ and $N(v)$ is the neighborhood of $v$

– for $n = 0$ to $h$
    1. Multiset-label determination
        – for each $v \in V$
            – if $n = 0$, $M_n(v) = l_0(v) = \ell(v)$
            – if $n > 0$, $M_n(v) = \{l_{n-1}(u) | u \in N(v)\}$
    2. Sorting each multiset
        – for each $M_n(v)$, sort the elements in $M_n(v)$, in ascending order and concatenate them into a string $s_n(v)$
        – for each $s_n(v)$, if $n > 0$, add $l_{n-1}(v)$ as a prefix to $s_n(v)$
    3. Label compression
        – for each $s_n(v)$
            – sort all strings $s_n(v)$ together in ascending order
            – map each string $s_n(v)$ to a new compressed label, using a function $f : \Sigma^* \rightarrow \Sigma$, such that $f(s_n(v)) = f(s_n(v'))$ iff $s_n(v) = s_n(v')$
    4. Relabeling
        – for each $s_n(v)$, set $l_n(v) = f(s_n(v))$
```

$$k^h_{\mathrm{WL}}(G, G') = \sum_{n=0}^{h} k_\delta(G_n, G'_n) \ , \tag{1}$$

where

$$k_\delta((V, E, l), (V', E', l')) = \sum_{v \in V} \sum_{v' \in V'} \delta(l(v), l'(v')) \ . \tag{2}$$

Here δ is the Dirac kernel, which tests for equality, it is 1 *if its arguments are equal, and* 0 *otherwise.*

Essentially this kernel counts the common vertex labels in each of the iterations of the graph rewriting process.

Instead of computing the Weisfeiler-Lehman relabeling on pairs of graphs, as in Algorithm 1, it can just as easily be computed on a set of graphs. Furthermore, the label dictionary f can be used to construct a feature vector for each graph. Then the kernel can be computed by taking the dot product of these feature vectors, which speeds up the computation of the kernel. See [4] for more details.

2.3 Fast Weisfeiler-Lehman for RDF

Since graph kernels compute a similarity between graphs, the most immediate approach to apply graph kernels to RDF is to extract subgraphs for the instances that we are interested in and to compute the kernel on these subgraphs. This approach is followed in [5] for most of the kernels. The intuition is that these subgraphs contain properties of the instances and that they say something about the position of the instances in the larger graph. However, the subgraphs are derived from the same underlying RDF graph and since they are instances of a similar concept they often have a number of vertices and edges in common. Potentially it can be more efficient to do the kernel computation directly on the larger underlying RDF graph.

In this section we introduce an approximation of the Weisfeiler-Lehman kernel designed for RDF data. We could just apply the Weisfeiler-Lehman relabeling as we have defined it above to the underlying RDF graph and then count the resulting labels in the neighborhood up to a certain depth for each instance vertex. However, by the nature of the relabeling process this means that vertices/edges on the border of the neighborhood are influenced by vertex/edges outside of the neighborhood. This means that the subgraph perspective for each instance is essentially gone, and that interesting information about the position of the instance in the graph is lost. We deal with this problem by tracking for each vertex and edge in the graph at what depth it occurs in the subgraphs of the instances.

First we define, in Definition 2, the type of graph that we apply the relabeling process to.

Definition 2. *A Weisfeiler-Lehman RDF graph is a graph $G = (V, E, l)$, where V is a set of vertices, E a set of directed edges, and $l : (V \cup E) \times \mathbb{N} \to \Sigma$ a labeling function from vertices V or edges E and a depth index $j \in \mathbb{N}$ to a set of labels Σ.*

This graph is a directed multigraph with a special labeling function, which gives the label for a vertex or edge given an index j. This index j indicates the depth at which this vertex or edge was encountered in the extraction of the subgraph.

We will also need our variant of neighborhoods of vertices and edges, as given in Definition 3.

Definition 3. *The neighborhood $N(v) = \{(v', v) \in E\}$ of a vertex is the set of edges going to the vertex v and the neighborhood $N((v, v')) = \{v\}$ of an edge is the vertex that the edge comes from.*

The graph extraction algorithm, given in Algorithm 2, creates an RDF graph, as given in Definition 2, for a set of instances I. For each instance i a subgraph up to depth d is extracted from the RDF dataset and this subgraph is added to the total graph G that the algorithm is building. Thus, vertices and edges are only added if they have not been added to the graph already (which is recorded using the *vMap* and *eMap* datastructures). For each vertex and edge encountered during the extraction process a label is saved (in ℓ) for the depth j at which the vertex or edge is encountered. For example, if a vertex v would only occur at depths 1 and 2 in all of the extracted subgraphs, then we would have $\ell(v, 1) = o$ and $\ell(v, 2) = o$. Next to the graph G we also construct mappings $\mathcal{V}_i$ and $\mathcal{E}_i$ for each instance i, which records which vertices and edges belong to the subgraph of instance i and at which depth.

Algorithm 2. Graph Creation from RDF

```
Input  a set of RDF triples $R$, a set of instances $I$ and extraction depth $d$
Output  a Weisfeiler-Lehman RDF graph $G = (V, E, \ell)$, mappings $\mathcal{V}_i$ from vertices to
    integers and $\mathcal{E}_i$ from edges to integers for each instance $i$.

1. Initialization
   – for each $i \in I$:
      – add a vertex $v$ to $V$, set $\ell(v, d) = \epsilon$ and set $vMap(i) = v$
2. Subgraph Extraction
   – for each $i \in I$:
      – $searchFront = \{i\}$
      – for $j = d - 1$ to $0$
         – $newSearchFront = \emptyset$
         – for each $r \in searchFront$:
            – $triples = \{(r, p, o) \in R\}$
            – for each $(s, p, o) \in triples$:
               – add $o$ to $newSearchFront$
               – if $vMap(o)$ is undefined, add vertex $v$ to $V$ and set $vMap(o) = v$
               – set $\ell(vMap(o), j) = o$
               – if $\mathcal{V}_i(vMap(o))$ is undefined, set $\mathcal{V}_i(vMap(o)) = j$
               – if $eMap(s, p, o)$ is undefined, add edge $e$ to $E$ and set
                 $eMap(s, p, o) = e$
               – set $\ell(eMap(s, p, o), j) = p$
               – if $\mathcal{E}_i(eMap(s, p, o))$ is undefined, set $\mathcal{E}_i(eMap(s, p, o)) = j$
         – $searchFront = newSearchFront$
```

Algorithm 3 describes the Weisfeiler-Lehman relabeling for graphs constructed using Algorithm 2. There are two main differences compared to the standard Weisfeiler-Lehman algorithm of Algorithm 1. The first difference is the extension to directed edges with labels, which is relatively straightforward. The second difference is in the construction of the multisets M_n, which are now constructed for a vertex/edge with a depth index j. For the vertices these multisets are constructed using the labels of the edges at depth $j - 1$, and for the edges with labels of the vertices of depth j. Furthermore, a vertex label at depth 0 is never rewritten.

Definition 4. *Let G be a Weisfeiler-Lehman RDF graph, created using Algorithm 2 and rewritten for h iterations using Algorithm 3, and l_0 to l_h the resulting label functions. Then we compute a kernel between two instances $i, i' \in I$, as:*

$$k^h_{\text{WLRDF}}(i, i') = \sum_{n=0}^{h} \frac{n+1}{h+1} k^n_{\delta,\text{RDF}}((\mathcal{V}_i, \mathcal{E}_i), (\mathcal{V}_{i'}, \mathcal{E}_{i'})), \tag{3}$$

where

Algorithm 3. Weisfeiler-Lehman Relabeling for RDF

```
Input a Weisfeiler-Lehman RDF graph $G = (V, E, \ell)$, subgraph depth $d$ and number
    of iterations $h$
Output label functions $l_0$ to $l_h$ and label dictionary $f$

– for $n = 0$ to $h$
    1. Multiset-label determination
        – for each $v \in V$ and $e \in E$ and $j = 0$ to $d$
            – if $n = 0$ and $\ell(v, j)$ is defined, set $M_n(v, j) = l_0(v, j) = \ell(v, j)$
            – if $n = 0$ and $\ell(e, j)$ is defined, set $M_n(e, j) = l_0(e, j) = \ell(e, j)$.
            – if $n > 0$ and $\ell(v, j)$ is defined, set $M_n(v, j) = \{l_{n-1}(u, j) | u \in N(v)\}$
            – if $n > 0$ and $\ell(e, j)$ is defined, set $M_n(e, j) = \{l_{n-1}(u, j + 1) | u \in N(e)\}$
    2. Sorting each multiset
        – for each $M_n(v, j)$ and $M_n(e, j)$, sort the elements in $M_n(v, j)$, resp. $M_n(e, j)$,
            in ascending order and concatenate them into a string $s_n(v, j)$, resp.
            $s_n(e, j)$
        – for each $s_n(v, j)$ and $s_n(e, j)$, if $n > 0$, add $l_{n-1}(v, j)$, resp. $l_{n-1}(e, j)$, as a
            prefix to $s_n(v, j)$, resp. $s_n(e, j)$
    3. Label compression
        – for each $s_n(v, j)$ and $s_n(e, j)$, map $s_n(v, j)$, resp. $s_n(e, j)$, to a new com-
            pressed label, using a function $f : \Sigma^* \to \Sigma$, such that $f(s_n(v, j)) =$
            $f(s_n(v', j))$ iff $s_n(v, j) = s_n(v', j)$, resp. $f(s_n(e, j)) = f(s_n(e', j))$ iff
            $s_n(e, j) = s_n(e', j)$
    4. Relabeling
        – for each $s_n(v, j)$ and $s_n(e, j)$, set $l_n(v, j) = f(s_n(v, j))$ and $l_n(e, j) =$
            $f(s_n(e, j))$
```

$$k^n_{\delta,\mathrm{RDF}}((\mathcal{V}_i, \mathcal{E}_i), (\mathcal{V}_{i'}, \mathcal{E}_{i'})) = \sum_{(v,d)\in\mathcal{V}_i} \sum_{(v',d')\in\mathcal{V}_{i'}} \delta(l_n(v, d), l_n(v', d')) + \sum_{(e,d)\in\mathcal{E}_i} \sum_{(e',d')\in\mathcal{E}_{i'}} \delta(l_n(e, d), l_n(e', d')) \ . \tag{4}$$

Here δ *is the Dirac kernel, which tests for equality, it is* 1 *if its arguments are equal, and* 0 *otherwise.*

This kernel is very similar to the regular definition of the Weisfeiler-Lehman Subtree kernel. Ones difference is that instances are not represented by their graphs but by the two maps $\mathcal{V}_i$ and $\mathcal{E}_i$. Furthermore, there is an added part to sum over all the edges. Like the regular Weisfeiler-Lehman Subtree kernel, this kernel is an instance of a convolution kernel [12]. We have added the factor $\frac{n+1}{h+1}$ to put more weight on higher iterations, to weigh more complicated structural similarity more heavily.

The resulting kernel k_{WLRDF} is an approximation of k_{WL} (provided that we add edge relabeling to the algorithm). Differences occur when there are cycles in the subgraph. For instance, let vertex v_1 have a label o_1 at depth 3 and v_1 has an edge to v_2. Vertex v_2 has a label o_2 at depth 2 and has an edge back to

v_1. Therefore v_1 will have a label o_3 at depth 1. During the relabeling, the label o_3 will (eventually) be combined with label o_2 and label o_2 will be combined with o_1 (not o_3, which seems more intuitive). In the regular Weisfeiler-Lehman variant, there would be no labels at different depths, so o_3 would be combined with o_2 and vice versa.

In [4] it is shown that the runtime for the relabeling algorithm on a set of graphs is $\mathcal{O}(Nhm)$, where N is the number of graphs, h is the number of iterations and m is the number of vertices (and edges) per graph. For our relabeling method we do not have N graphs, but we do introduce depth d labels per vertex/edge. Our larger graph has a number of vertices and edges k. Since our algorithm is essentially regular Weisfeiler-Lehman with the addition of multiple labels per vertex/edge, the runtime complexity for our relabeling algorithm is $\mathcal{O}(hkd)$. As for the regular Weisfeiler-Lehman Subtree kernel we can create feature vectors for each instance, using the label dictionary f. Hence, in situations with $kd < Nm$, our algorithm will be faster. This scenario is typical for the RDF use-case, where the subgraphs for each instance share a (large) number of vertices and edges, which means $Nm \gg k$ given large enough N, and therefore $kd < Nm$ if $d \ll N$. The same bounds hold for the space complexity as for the runtime complexity. Therefore, in situations with $kd < Nm$, our algorithm requires less memory than regular Weisfeiler-Lehman on a set of graphs.

3 Experiments

In this section we present a number of experiments with the Weisfeiler-Lehman for RDF (WL RDF) kernel presented above. The goal of these experiments is to compare the prediction performance of this kernel to the regular Weisfeiler-Lehman (WL) kernel, adapted to handle edge labels and using the same iteration weighting. For comparison we use three prediction tasks using RDF data. Since the WL RDF kernel is intended to be a faster variant of the WL kernel we also compare the runtimes.

Furthermore, we compare the WL RDF kernel with the kernels designed for RDF in [5]. These kernels are the efficient Intersection SubTree (IST) and Intersection Partial SubTree (IPST) kernels and the inefficient Intersection Graph Walk (IGW) and Intersection Graph Path (IGP) kernels. The intersection subtree kernels are based on counting the number of (partial) subtrees in the intersection tree of two graphs. The intersection graph kernels count the number of paths/walks in the intersection graph of two graphs.

Like we did for the WL RDF kernel, [5] compute the IST and IPST kernels directly on the RDF graph. For the WL, IGW and IGP kernels subgraphs have to be extracted. For each kernel we test 3 extraction depths (1,2,3) and we also test with and without RDFS inferencing by the triple-store. RDFS inferencing potentially derives new triples based on logical relations between the concepts defined in the RDF. We test these different settings to see the influence of larger subgraphs on the prediction performance.

All of the kernels and experiments where implemented in Java and the code is available online.[3] The Java version of the LibSVM [13] support vector machine library was used for prediction with the kernels and the SESAME[4] triple-store was used to handle the RDF data and do the RDFS inferencing. The experiments where run on an AMD X6 1090T CPU with 16 GB RAM.

3.1 Affiliation Prediction

For our first experiment we repeat the affiliation prediction task introduced in [6] and repeated in [5]. This experiment uses data of the semantic portal of the AIFB research institute modeled in the SWRC ontology [14], which models key concepts in a research community. The data contains 178 persons that belong to 1 of 5 research institutes. Furthermore it contains information about publications, students, etc. One institute contains only 4 members, which we ignore. The goal of the prediction task is to predict the affiliation for the remaining 174 instances. Since we know the affiliations, for training purposes the affiliation relation (and its inverse the employs relation) are removed from the RDF for each instance. Also we set the label of the root vertex for each instance to an identical special root label (like in [5]), since the original URI is unique for each instance.

For the Weisfeiler-Lehman kernels we test the h settings: $0, 2, 4, 6$. The two intersection graph kernels have a maximum path length parameter, which we also call h, for which we test $1, 2$.[5] All the four kernels from [5] have a discount factor parameter λ and are tested with the setting reported to give the best results.

For each kernel we use the C-SVC support vector machine algorithm from LibSVM to train a classifier to predict the affiliation. Per kernel we do a 10-fold cross-validation which is repeated 10 times. Within each fold the C parameter is optimized from the range: $\{10^{-3}, 10^{-2}, 0.1, 1, 10, 10^2, 10^3\}$ by doing 10-fold cross-validation. We also weigh the different classes with the inverse of their frequency. All kernels are normalized.

Table 1 presents the average accuracy and F1[6] scores. The best scores, and the scores that have no significant difference with these scores under a Student t-test with $p < 0.05$, are indicated using a bold type face.

The best performance is achieved by our Weisfeiler-Lehman RDF kernel variant, showing slightly better scores than regular Weisfeiler-Lehman in the '3,f' setting. The performance of the intersection graph kernels comes close to the WL kernels, but the intersection tree kernels clearly show worse performance. Increasing extraction depth increases performance for all the tested kernels but the intersection trees. Adding inferencing only benefits the WL-kernels.

The Weisfeiler-Lehman kernel under the $h = 0$ setting can be considered as a baseline method, because it is essentially a 'bag-of-labels' kernel, where

[3] `https://github.com/Data2Semantics/d2s-tools`

[4] `http://www.openrdf.org/`

[5] Higher settings take a very large amount of computation time and/or run out of memory.

[6] This is the average of the F1 scores for each class.

Table 1. Results for the affiliation prediction experiments. 1,2,3 indicate the subgraph depth and 'f' indicates that inferencing was applied.

	acc.	F1	acc.	F1	acc.	F1	acc.	F1
	Weisfeiler-Lehman RDF							
	$h = 0$		$h = 2$		$h = 4$		$h = 6$	
1	0.84	0.67	0.84	0.67	0.84	0.67	0.84	0.67
2	0.83	0.66	0.87	0.72	0.86	0.70	0.86	0.70
3	0.85	0.71	0.89	0.79	0.89	0.77	0.88	0.76
1,f	0.79	0.59	0.79	0.59	0.79	0.59	0.79	0.59
2,f	0.57	0.35	0.84	0.66	0.81	0.62	0.81	0.61
3,f	0.73	0.56	**0.91**	**0.81**	0.90	**0.80**	0.90	0.79
	IntersectionSubTree, $\lambda = 1$				IntersectionPartialSubTree, $\lambda = 0.01$			
1	0.83	0.64			0.81	0.61		
2	0.82	0.61			0.81	0.61		
3	0.82	0.61			0.79	0.60		
1,f	0.81	0.61			0.79	0.58		
2,f	0.79	0.58			0.78	0.58		
3,f	0.81	0.61			0.78	0.58		
	Weisfeiler-Lehman							
	$h = 0$		$h = 2$		$h = 4$		$h = 6$	
1	0.83	0.66	0.84	0.67	0.84	0.67	0.84	0.67
2	0.87	0.74	0.84	0.67	0.78	0.54	0.75	0.48
3	0.86	0.72	0.88	0.77	0.88	0.75	0.86	0.71
1,f	0.79	0.60	0.79	0.60	0.79	0.60	0.79	0.60
2,f	0.58	0.36	0.83	0.64	0.79	0.57	0.73	0.47
3,f	0.73	0.56	0.89	0.78	0.89	0.77	0.87	0.72
	IntersectionGraphPath, $\lambda = 1$				IntersectionGraphWalk, $\lambda = 1$			
	$h = 1$		$h = 2$		$h = 1$		$h = 2$	
1	0.84	0.65	0.84	0.65	0.84	0.64	0.84	0.64
2	0.82	0.61	0.80	0.59	0.82	0.61	0.80	0.59
3	0.88	0.76	0.90	0.77	0.89	0.76	0.89	0.77
1,f	0.81	0.61	0.81	0.61	0.81	0.61	0.81	0.61
2,f	0.79	0.58	0.75	0.51	0.79	0.58	0.71	0.48
3,f	0.88	0.76	0.88	0.76	0.88	0.76	0.88	0.75

no rewriting is performed. We can see that already quite good performance is achieved using this baseline.

3.2 Lithogenesis Prediction

We perform the next prediction experiment on the RDF dataset from the British Geological Survey[7], which contains information about geological measurements in Britain. This dataset was chosen because it is at least a factor 10 larger than the affiliation prediction set and has some potential nice properties to predict. The things that are measured by this survey are called 'Named Rock Units', which have a number of different properties. One of these is the lithogenesis property, for which the two largest classes have 93 and 53 instances. In this experiment we try to predict for these 146 instances which of these two classes it belongs too. Again we remove triples related to these properties from the dataset and set the labels of the root vertices to the same special root label.

[7] `http://data.bgs.ac.uk/`

The setup for this experiment is the same as for the affiliation prediction task. The results are presented in Table 2.

The results of this experiment are similar to the results for affiliation prediction. Again the best scores are achieved by the WL RDF kernel under the '3,f' setting. However, the intersection graph path kernel achieves a similar accuracy score and the intersection subtree kernel scores are closer to WL RDF kernel. Increasing the subgraph depth improves the performance for all kernels. The performance of the 'bag-of-labels' baseline is similar to the affiliation prediction task.

3.3 Runtimes

To test the differences in runtimes between the kernels we measure the runtimes for the computation of each of the kernels on the two datasets above under the highest extraction setting (depth 3 and inferencing on). We measure these runtimes for different fractions of the dataset, from 0.1 to 1. The computation times of the two intersection tree kernels are nearly identical, so we only include

Table 2. Results for the lithogenesis prediction experiments. 1,2,3 indicate the subgraph depth and 'f' indicates that inferencing was applied.

	acc.	F1	acc.	F1	acc.	F1	acc.	F1
	Weisfeiler-Lehman RDF							
	$h = 0$		$h = 2$		$h = 4$		$h = 6$	
1	0.79	0.62	0.79	0.62	0.79	0.62	0.79	0.62
2	0.87	0.75	0.88	0.77	0.88	0.76	0.88	0.77
3	0.86	0.73	0.87	0.75	0.88	0.76	0.88	0.77
1,f	0.78	0.61	0.78	0.61	0.78	0.61	0.78	0.61
2,f	0.82	0.66	0.88	0.76	0.87	0.75	0.87	0.75
3,f	0.88	0.75	0.89	0.78	**0.91**	**0.82**	**0.91**	**0.82**
	IntersectionSubTree, $\lambda = 1$				IntersectionPartialSubTree, $\lambda = 0.01$			
1	0.79	0.63			0.81	0.65		
2	0.85	0.71			0.82	0.66		
3	0.86	0.73			0.82	0.67		
1,f	0.78	0.60			0.79	0.62		
2,f	0.84	0.70			0.80	0.63		
3,f	0.85	0.72			0.80	0.64		
	Weisfeiler-Lehman							
	$h = 0$		$h = 2$		$h = 4$		$h = 6$	
1	0.79	0.62	0.79	0.62	0.79	0.63	0.79	0.63
2	0.88	0.77	0.86	0.73	0.85	0.70	0.84	0.69
3	0.86	0.73	0.87	0.75	0.87	0.75	0.88	0.76
1,f	0.78	0.61	0.78	0.61	0.78	0.61	0.78	0.61
2,f	0.82	0.65	0.85	0.72	0.85	0.71	0.85	0.71
3,f	0.88	0.75	0.88	0.77	0.88	0.76	0.88	0.77
	IntersectionGraphPath, $\lambda = 1$				IntersectionGraphWalk, $\lambda = 1$			
	$h = 1$		$h = 2$		$h = 1$		$h = 2$	
1	0.81	0.66	0.82	0.67	0.79	0.62	0.80	0.63
2	0.86	0.72	0.86	0.72	0.86	0.73	0.86	0.72
3	0.88	0.76	0.88	0.76	0.88	0.76	0.87	0.75
1,f	0.80	0.64	0.81	0.64	0.77	0.60	0.77	0.60
2,f	0.85	0.71	0.85	0.72	0.84	0.70	0.85	0.71
3,f	**0.90**	0.80	0.90	0.79	0.90	0.79	0.89	0.78

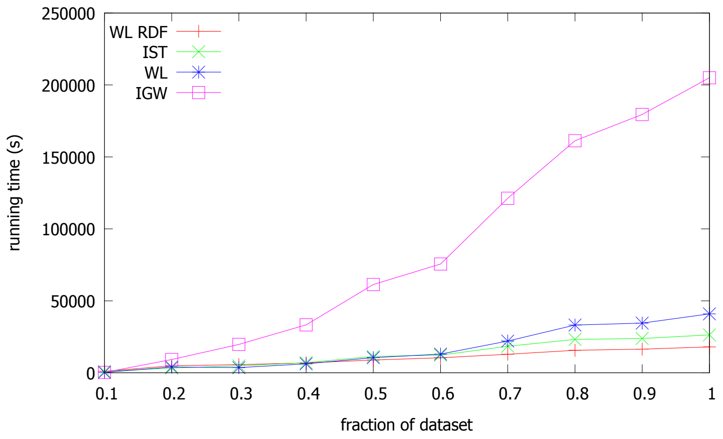

Fig. 1. Runtimes for the kernels: Weisfeiler-Lehman for RDF (WL RDF), Intersection SubTree (IST), Weisfeiler-Lehman (WL) and Intersection Graph Walk (IGW), for different fractions of the affiliation prediction dataset.

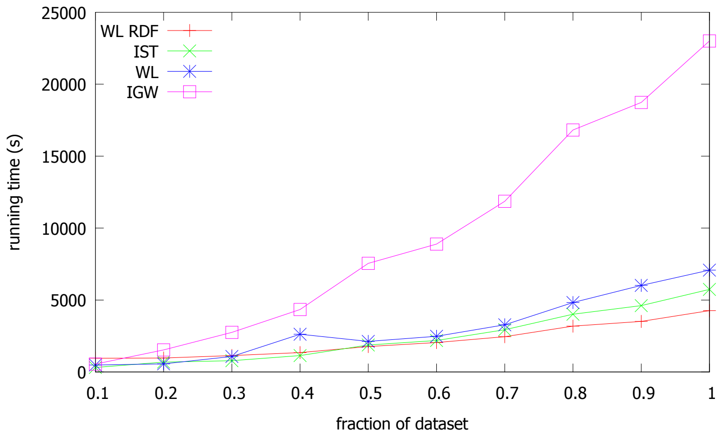

Fig. 2. Runtimes for the kernels: Weisfeiler-Lehman for RDF (WL RDF), Intersection SubTree (IST), Weisfeiler-Lehman (WL) and Intersection Graph Walk (IGW), for different fractions of the lithogenesis prediction dataset.

the IST kernel. The same is true for the two intersection graph kernels, so we only include the IGW kernel. The intersection subtree kernel is implemented as described in [5]. For the regular WL kernel and IGW kernel, the extraction of the subgraphs was also included, however this was only a small factor in the overall computation time.

Figure 1 presents the results for the four kernels on the affiliation prediction dataset. The results for the lithogenesis dataset are presented in Fig. 2.

Both figures show similar results: the IGW kernel is the slowest by a large margin and as the datasets grow larger, the WL RDF kernel becomes more efficient. The differences in runtimes between the WL RDF kernel and regular WL are in line with our expectations: as the amount of instances N become larger, WL RDF becomes more efficient on RDF datasets than regular WL.

3.4 Geological Theme Prediction

For the two most efficient kernel types, WL RDF and IST/IPST, we performed another experiment on the British Geological Survey data. All of the 'Named Rock Units', around 12000 instances, have an associated geological theme. In this experiment we try to predict that theme, which has two major classes, one with 10020 instances and the other with 1377. We try to predict whether an instance belongs to one of these two classes.

The setup for this experiment is similar to the two prediction tasks presented above. However, we do not repeat the experiment 10 times for the full dataset, but we take 10 random 10% subsets of the full dataset. The results are presented in Table 3. Again bold type face indicates the best scores. This time a MannWhitney U test with $p < 0.05$ was used as a significance test, since the resulting scores did not fit a normal distribution.

As in the previous experiments, the Weisfeiler-Lehman RDF kernel shows the best performance. Almost perfect scores are achieved. However, the intersection subtree kernels and the 'bag-of-labels' baseline come very close to this performance. For the tree kernels it holds that increasing the extraction depth increases performance.

3.5 No Labels

The 'bag-of-labels' baseline, shows already good performance in the three tasks. To test whether this is due to the fact that the graph structure provides little information and to see if the graph kernels can exploit structure information, we

Table 3. Results for the theme prediction experiments. 1,2,3 indicate the subgraph depth and 'f' indicates that inferencing was applied.

	acc.	F1	acc.	F1	acc.	F1	acc.	F1
	Weisfeiler-Lehman RDF							
	$h = 0$		$h = 2$		$h = 4$		$h = 6$	
1	0.90	0.69	0.93	0.79	0.96	0.85	0.96	0.84
2	0.94	0.78	0.97	0.89	0.95	0.85	0.97	0.91
3	0.98	0.89	**1.0**	**0.98**	**1.0**	**0.98**	**0.99**	**0.98**
1,f	0.88	0.66	0.94	0.79	0.96	0.84	0.96	0.85
2,f	0.88	0.65	0.92	0.79	0.98	0.91	0.95	0.88
3,f	0.74	0.44	**1.0**	**0.98**	**0.99**	**0.98**	**1.0**	**0.98**
	IntersectionSubTree, $\lambda = 1$				IntersectionPartialSubTree, $\lambda = 0.01$			
1	0.93	0.79			0.88	0.68		
2	0.96	0.89			0.97	0.88		
3	0.98	0.93			0.97	0.88		
1,f	0.94	0.84			0.94	0.78		
2,f	0.97	0.91			0.97	0.86		
3,f	0.99	0.96			0.98	0.92		

Table 4. Results for the affiliation prediction experiments with vertex and edge labels removed. 1,2,3 indicate the subgraph depth and 'f' indicates that inferencing was applied.

	acc.	F1	acc.	F1	acc.	F1	acc.	F1
	Weisfeiler-Lehman RDF							
	$h = 0$		$h = 2$		$h = 4$		$h = 6$	
1	0.14	0.07	0.61	0.40	0.63	0.41	0.63	0.41
2	0.48	0.23	0.61	0.40	0.62	0.42	0.64	0.44
3	0.43	0.24	0.87	0.75	**0.89**	**0.77**	0.88	0.75
1,f	0.11	0.06	0.51	0.31	0.51	0.31	0.51	0.31
2,f	0.26	0.12	0.57	0.36	0.60	0.39	0.61	0.40
3,f	0.18	0.09	0.85	0.70	0.87	0.74	0.88	0.75
	IntersectionSubTree, $\lambda = 1$				IntersectionPartialSubTree, $\lambda = 0.01$			
1	0.11	0.06			0.11	0.06		
2	0.22	0.09			0.11	0.06		
3	0.23	0.10			0.16	0.07		
1,f	0.11	0.06			0.11	0.06		
2,f	0.23	0.10			0.12	0.06		
3,f	0.23	0.10			0.25	0.09		
	Weisfeiler-Lehman							
	$h = 0$		$h = 2$		$h = 4$		$h = 6$	
1	0.14	0.07	0.50	0.24	0.51	0.25	0.50	0.24
2	0.48	0.23	0.47	0.25	0.46	0.24	0.46	0.24
3	0.43	0.24	0.59	0.37	0.72	0.51	0.72	0.49
1,f	0.11	0.06	0.32	0.17	0.33	0.17	0.33	0.17
2,f	0.29	0.15	0.53	0.33	0.51	0.29	0.52	0.29
3,f	0.16	0.08	0.57	0.37	0.73	0.54	0.71	0.50
	IntersectionGraphPath, $\lambda = 1$				IntersectionGraphWalk, $\lambda = 1$			
	$h = 1$		$h = 2$		$h = 1$		$h = 2$	
1	0.11	0.06	0.11	0.06	0.45	0.26	0.46	0.26
2	0.48	0.20	0.44	0.24	0.47	0.26	0.36	0.20
3	0.48	0.20	0.45	0.24	0.55	0.34	0.52	0.31
1,f	0.11	0.06	0.11	0.06	0.47	0.27	0.46	0.26
2,f	0.42	0.18	0.45	0.26	0.47	0.27	0.38	0.19
3,f	0.33	0.16	0.45	0.26	0.51	0.31	0.33	0.16

repeat the affiliation prediction experiment. This time we remove all the label information after the creation of the graph/subgraphs. Each label is replaced by the same dummy label. The rest of the experimental setup is identical to the affiliation prediction experiment. Results are given in Table 4.

Again, the Weisfeiler-Lehman RDF kernel shows the best performance. What is even more striking is that a performance close to the performance on labeled graphs can be achieved. The regular WL kernel also does reasonably well, however, the intersection graph and the intersection subtree kernel show very poor performance. These kernels are clearly not designed for unlabeled graphs and do not exploit graph structure as the WL kernels do.

4 Conclusions and Future Work

We presented an approximation of the Weisfeiler-Lehman Subtree kernel which speeds up computation on RDF graphs by computing the kernel on the underlying RDF graph instead of extracted subgraphs. The kernel shows performance that is better than the regular Weisfeiler-Lehman kernel applied to RDF. Also it is increasingly more efficient as the number of instances grows. This efficiency is achieved by exploiting the fact that the RDF instance subgraphs share vertices and edges in the underlying large RDF graph.

Furthermore, the presented kernel is faster and shows better classification performance than the intersection subtree and intersection graph kernels, which were recently introduced specifically for RDF data. When we remove the label data, then the Weisfeiler-Lehman for RDF still has relatively good classification performance on just the graph structure, whereas the intersection subtree and intersection graph kernels cannot use the structure information very well.

The performance difference between the presented approximation of the WL Subtree kernel and the regular version requires further investigation. However, we have observed that the computed feature vectors for WL RDF are shorter than for regular WL. Thus, our approximation probably leads to more shared features between instances, which can result in better generalization and less overfitting.

The presented kernel can be used in any machine learning situation where subgraphs derive from an underlying larger graph. It is particularly well-suited to RDF because the extracted subgraphs share a large number of vertices and edges. As future work it would be interesting to apply the kernel to other similar situations.

Another direction for future research is the application of the presented kernel to extremely large RDF datasets with 100 millions of triples or more, since it has the potential to scale well. For datasets with more instances, the computed feature vectors can be used directly with large scale linear SVM methods, such as LibLINEAR [15], which is not possible with the intersection subtree and intersection graph kernels. We also wish to investigate extensions of the Weisfeiler-Lehman kernel using label comparisons other the than the Dirac-kernel.

Acknowledgments. This publication was supported by the Dutch national program COMMIT. We express our gratitude to the anonymous reviewers for their helpful comments. We thank the authors of [5] for the AIFB dataset.

References

1. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, New York (2004)
2. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2001)
3. Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R.I., Borgwardt, K.M.: Graph kernels. Journal of Machine Learning Research 11, 1201–1242 (2010)
4. Shervashidze, N., Schweitzer, P., van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. J. Mach. Learn. Res. 12, 2539–2561 (2011)
5. Lösch, U., Bloehdorn, S., Rettinger, A.: Graph kernels for RDF data. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 134–148. Springer, Heidelberg (2012)

6. Bloehdorn, S., Sure, Y.: Kernel methods for mining instance data in ontologies. In: Aberer, K., et al. (eds.) ISWC/ASWC 2007. LNCS, vol. 4825, pp. 58–71. Springer, Heidelberg (2007)
7. Bicer, V., Tran, T., Gossen, A.: Relational kernel machines for learning from graph-structured RDF data. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 47–62. Springer, Heidelberg (2011)
8. Fanizzi, N., d'Amato, C.: A declarative kernel for $\mathcal{ALC}$ concept descriptions. In: Esposito, F., Raś, Z.W., Malerba, D., Semeraro, G. (eds.) ISMIS 2006. LNCS (LNAI), vol. 4203, pp. 322–331. Springer, Heidelberg (2006)
9. Fanizzi, N., d'Amato, C., Esposito, F.: Statistical learning for inductive query answering on OWL ontologies. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 195–212. Springer, Heidelberg (2008)
10. Rettinger, A., Nickles, M., Tresp, V.: Statistical relational learning with formal ontologies. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009, Part II. LNCS, vol. 5782, pp. 286–301. Springer, Heidelberg (2009)
11. Shervashidze, N., Borgwardt, K.M.: Fast subtree kernels on graphs. In: Bengio, Y., Schuurmans, D., Lafferty, J.D., Williams, C.K.I., Culotta, A. (eds.) NIPS, pp. 1660–1668. Curran Associates, Inc. (2009)
12. Haussler, D.: Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA (1999)
13. Chang, C.C., Lin, C.J.: 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`
14. Sure, Y., Bloehdorn, S., Haase, P., Hartmann, J., Oberle, D.: The SWRC ontology - semantic web for research communities. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS (LNAI), vol. 3808, pp. 218–231. Springer, Heidelberg (2005)
15. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (2008)