kLogNLP: Graph Kernel–based Relational Learning of Natural Language

Mathias Verbeke[◇] **Paolo Frasconi**[♠] **Kurt De Grave**[◇] **Fabrizio Costa**[♣] **Luc De Raedt**[◇]

[◇] Department of Computer Science, KU Leuven, Belgium
{mathias.verbeke, kurt.degrave, luc.deraedt}@cs.kuleuven.be
[♠] Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Italy,
p-f@dsi.unifi.it
[♣] Institut für Informatik, Albert-Ludwigs-Universität, Germany,
costa@informatik.uni-freiburg.de

Abstract

kLog is a framework for kernel-based learning that has already proven successful in solving a number of relational tasks in natural language processing. In this paper, we present *kLogNLP*, a natural language processing module for kLog. This module enriches kLog with NLP-specific preprocessors, enabling the use of existing libraries and toolkits within an elegant and powerful declarative machine learning framework. The resulting relational model of the domain can be extended by specifying additional relational features in a declarative way using a logic programming language. This declarative approach offers a flexible way of experimentation and a way to insert domain knowledge.

1 Introduction

kLog (Frasconi et al., 2012) is a logical and relational language for kernel-based learning. It has already proven successful for several tasks in computer vision (Antanas et al., 2012; Antanas et al., 2013) and natural language processing. For example, in the case of binary sentence classification, we have shown an increase of 1.2 percent in F1-score on the best performing system in the CoNLL 2010 Shared Task on hedge cue detection (Wikipedia dataset) (Verbeke et al., 2012a). On a sentence labeling task for evidence-based medicine, a multi-class multi-label classification problem, kLog showed improved results over both the state-of-the-art CRF-based system of Kim et al. (2011) and a memory-based benchmark (Verbeke et al., 2012b). Also for spatial relation extraction from natural language, kLog has shown to provide a flexible relational representation to model the task domain (Kordjamshidi et al., 2012).

kLog has two distinguishing features. First, it is able to transform relational into graph-based representations, which allows to incorporate structural features into the learning process. Subsequently, kernel methods are used to work in an extended high-dimensional feature space, which is much richer than most of the direct propositionalisation approaches. Second, it uses the logic programming language Prolog for defining and using (additional) background knowledge, which renders the model very interpretable and provides more insights into the importance of individual (structural) features.

These properties prove especially advantageous in the case of NLP. The graphical approach of kLog is able to exploit the full relational representation that is often a natural way to express language structures, and in this way allows to fully exploit contextual features. On top of this relational learning approach, the declarative feature specification allows to include additional background knowledge, which is often essential for solving NLP problems.

In this paper, we present *kLogNLP*[1], an NLP module for kLog. Starting from a dataset and a declaratively specified model of the domain (based on entity-relationship modeling from database theory), it transforms the dataset into a graph-based relational format. We propose a general model that fits most tasks in NLP, which can be extended by specifying additional relational features in a declarative way. The resulting relational representation then serves as input for kLog, and thus results in a full relational learning pipeline for NLP.

kLogNLP is most related to Learning-Based Java (LBJ) (Rizzolo and Roth, 2010) in that it offers a declarative pipeline for modeling and learning tasks in NLP. The aims are similar, namely abstracting away the technical details from the programmer, and leaving him to reason about the modeling. However, whereas LBJ focuses more on the learning side (by the specification of constraints on features which are reconciled at inference time, using the constrained conditional

[1] Software available at http://dtai.cs.kuleuven.be/klognlp

Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 85–90,
Baltimore, Maryland USA, June 23-24, 2014. ©2014 Association for Computational Linguistics

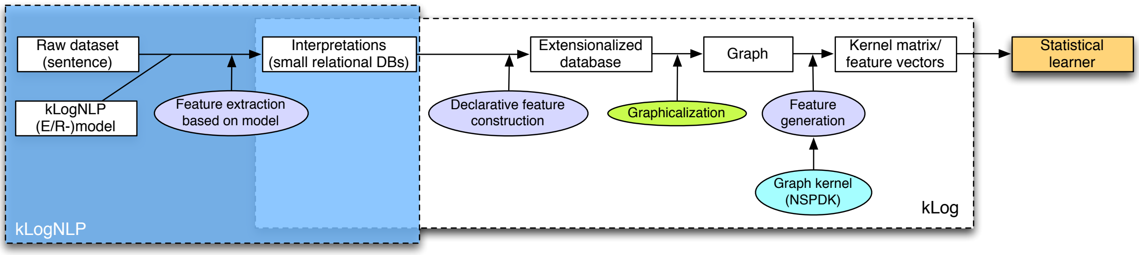

Figure 1: General kLog workflow extended with the kLogNLP module

model framework), due to its embedding in kLog, kLogNLP focuses on the relational modeling, in addition to declarative feature construction and feature generation using graph kernels. kLog in itself is related to several frameworks for relational learning, for which we refer the reader to (Frasconi et al., 2012).

The remainder of this paper is organized according to the general kLog workflow, preceded with the kLogNLP module, as outlined in Figure 1. In Section 2, we discuss the modeling of the data, and present a general relational data model for NLP tasks. Also the option to declaratively construct new features using logic programming is outlined. In the subsequent parts, we will illustrate the remaining steps in the kLog pipeline, namely graphicalization and feature generation (Section 3), and learning (Section 4) in an NLP setting. The last section draws conclusions and presents ideas for future work.

2 Data Modeling

kLog employs a learning from interpretations setting (De Raedt et al., 2008). In learning from interpretations, each interpretation is a set of tuples that are true in the example, and can be seen as a small relational database. Listing 3, to be discussed later, shows a concise example. In the NLP setting, an interpretation most commonly corresponds to a document or a sentence. The scope of an interpretation is either determined by the task (e.g., for document classification, the interpretations will at least need to comprise a single document), or by the amount of context that is taken into account (e.g., in case the task is sentence classification, the interpretation can either be a single sentence, or a full document, depending on the scope of the context that you want to take into account).

Since kLog is rooted in database theory, the modeling of the problem domain is done using an entity-relationship (E/R) model (Chen, 1976). It gives an abstract representation of the interpretations. E/R models can be seen as a tool that is tailored to model the domain at hand. As the name indicates, E/R models consist of entities, which we will represent as purple rectangles, and relations, represented as orange diamonds. Both entities and relations can have several attributes (yellow ovals). Key attributes (green ovals) uniquely identify an instance of an entity. We will now discuss the E/R model we propose as a starting point in the kLogNLP pipeline.

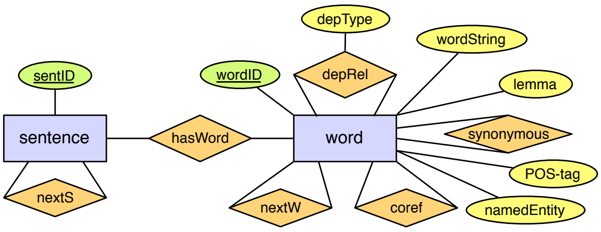

Figure 2: Entity-relationship diagram of the kLogNLP model

2.1 kLogNLP model

Since in NLP, most tasks are situated at either the document, sentence, or token level, we propose the E/R model in Figure 2 as a general domain model suitable for most settings. It is able to represent interpretations of documents as a sequence (`nextS`) of `sentence` entities, which are composed of a sequence (`nextW`) of `word` entities. Next to the sequence relations, also the dependency relations between words (`depRel`) are taken into account, where each relation has its type (`depType`) as a property. Furthermore, also the coreference relationship between words or phrases (`coref`) and possibly synonymy relations (`synonymous`) are taken into account. The entities in our model also have a primary key, namely `wordID` and `sentID` for words and sentences respectively. Additional properties can be attached to words such as the `wordString` itself, its `lemma` and `POS-tag`, and an indication whether the word is a `namedEntity`.

This E/R model of Figure 2 is coded declaratively in kLog as shown in Listing 1. The kLog syntax is an extension of the logical programming language Prolog. In the next step this script will be used for feature extraction and generation. Every entity or relationship is declared with the keyword `signature`. Each signature is of a certain type; either `extensional` or `intensional`. kLogNLP only acts at the extensional level. Each signature is characterized by a name and a list of typed arguments. There are three possible argument types. First of all, the type can be the name of an entity set which has been declared in another signature (e.g., line 4 in Listing 1; the `nextS` signature represents the sequence relation between two entities of type `sentence`, namely `sent_1` and `sent_2`). The type `self` is used to denote the primary key of an entity. An example is `word_id` (line 6), which denotes the unique identifier of a certain word in the interpretation. The last possible type is `property`, in case the argument is neither a reference to another entity nor a primary key (e.g., `postag`, line 9).

We will first discuss extensional signatures, and the automated extensional feature extraction provided by kLogNLP, before illustrating how the user can further enrich the model with intensional predicates.

```
1  begin_domain.
2  signature sentence(sent_id::self)::
       extensional.
3
4  signature nextS(sent_1::sentence, sent_2
       ::sentence)::extensional.
5
6  signature word(word_id::self,
7            word_string::property,
8            lemma::property,
9            postag::property,
10           namedentity::property
11           )::extensional.
12
13 signature nextW(word_1::word, word_2::
       word)::extensional.
14
15 signature corefPhrase(coref_id::self)::
       extensional.
16 signature isPartOfCorefPhrase(
       coref_phrase::corefPhrase, word::
       word)::extensional.
17 signature coref(coref_phrase_1::
       corefPhrase, coref_phrase_2::
       corefPhrase)::extensional.
18
19 signature synonymous(word_1::word,
       word_2::word)::extensional.
20
21 signature dependency(word_1::word,
22            word_2::word,
23            dep_rel::property
24           )::extensional.
25
26 kernel_points([word]).
27 end_domain.
```

Listing 1: Declarative representation of the kLogNLP model

2.2 Extensional Feature Extraction

kLog assumes a closed-world, which means that atoms that are not known to be true, are assumed to be false. For extensional signatures, this entails that all ground atoms need to be listed explicitly in the relational database of interpretations. These atoms are generated automatically by the kLogNLP module based on the kLog script and the input dataset. Considering the defined attributes and relations in the model presented in Listing 1, the module interfaces with NLP toolkits to preprocess the data to the relational format. The user can remove unnecessary extensional signatures or modify the number of attributes given in the standard kLogNLP script as given in Listing 1 according to the needs of the task under consideration.

An important choice is the inclusion of the `sentence` signature. By inclusion, the granularity of the interpretation is set to the document level, which implies that more context can be taken into account. By excluding this signature, the granularity of the interpretation is set to the sentence level.

Currently, kLogNLP interfaces with the following NLP toolkits:

NLTK The Python Natural Language Toolkit (NLTK) (Bird et al., 2009) offers a suite of text processing libraries for tokenization, stemming, tagging and parsing, and offers an interface to WordNet.

Stanford CoreNLP Stanford CoreNLP[2] provides POS tagging, NER, parsing and coreference resolution functionality.

The preprocessing toolkit to be used can be set using the kLogNLP flags mechanism, as illustrated by line 3 of Listing 2. Subsequently, the `dataset` predicate (illustrated in line 4 of Listing 2) calls kLogNLP to preprocess a given dataset[3]. This is done according to the specified kLogNLP model, i.e., the necessary preprocessing modules to be called in the preprocessing toolkit are determined based on the presence of the entities, relationships, and their attributes in the kLogNLP script. For example, the presence

[2] `http://nlp.stanford.edu/software/corenlp.shtml`

[3] Currently supported dataset formats are directories consisting of (one or more) plain text files or XML files consisting of sentence and/or document elements.

of `namedentity` as a property of `word` results in the addition of a named entity recognizer in the preprocessing toolkit. The resulting set of interpretations is output to a given file. In case several instantiations of a preprocessing module are available in the toolkit, the preferred one can be chosen by setting the name of the property accordingly. The names as given in Listing 1 outline the standard settings for each module. For instance, in case the Snowball stemmer is preferred above the standard (Wordnet) lemmatizer in NLTK, it can be selected by changing `lemma` into `snowball` as name for the word lemma property (line 8).

```
experiment :-
    % kLogNLP
    klognlp_flag(preprocessor,
        stanfordnlp),
    dataset('/home/hedgecuedetection/
        train/','trainingset.pl'),
    attach('trainingset.pl'),
    % Kernel parametrization
    new_feature_generator(my_fg,nspdk),
    klog_flag(my_fg,radius,1),
    klog_flag(my_fg,distance,1),
    klog_flag(my_fg,match_type, hard),
    % Learner parametrization
    new_model(my_model,libsvm_c_svc),
    klog_flag(my_model,c,0.1),
    kfold(target, 10, my_model, my_fg).
```

Listing 2: Full predicate for 10-fold classification experiment

Each interpretation can be regarded as a small relational database. We will illustrate the extensional feature extraction step on the CoNLL-2010 dataset on hedge cue detection, a binary classification task where the goal is to detect uncertainty in sentences. This task is situated at the sentence level, so we left out the `sentence` and `nextS` signatures, as no context from other sentences was taken into account. A part of a resulting interpretation is shown in Listing 3.

```
word(w1,often,often,rb,0,1).
depRel(w1,w5,adv).
nextW(w1,w2).
word(w2,the,the,dt,0,2).
depRel(w2,w4,nmod).
nextW(w2,w3).
word(w3,response,response,nn,0,3).
nextW(w3,w4).
depRel(w3,w4,nmod).
word(w4,may,may,md,0,5).
nextW(w4,w5).
```

Listing 3: Part of an interpretation

Optionally, additional extensional signatures can easily be added to the knowledge base by the user, as deemed suitable for the task under consideration. At each level of granularity (document, sentence, or word level), the user is given the corresponding interpretation and entity IDs, with which additional extensional facts can be added using the dedicated Python classes. We will now turn to declarative feature construction. The following steps are inherently part of the kLog framework. We will briefly illustrate their use in the context of NLP.

2.3 Declarative Feature Construction

The kLog script presented in Listing 1 can now be extended using declarative feature construction with *intensional* signatures. In contrast to extensional signatures, intensional signatures introduce novel relations using a mechanism resembling deductive databases. This type of signatures is mostly used to add domain knowledge about the task at hand. The ground atoms are defined implicitly using Prolog definite clauses.

For example, in case of sentence labeling for evidence-based medicine, the lemma of the root word proved to be a distinguishing feature (Verbeke et al., 2012b), which can be expressed as

```
signature lemmaRoot(sent_id::sentence,
    lemmaOfRoot::property)::intensional.
lemmaRoot(S,L) :-
   hasWord(S, I),
   word(I,_,L,_,_,_),
   depRel(I,_,root).
```

Also more complex features can be constructed. For example, section headers in documents (again in the case of sentence labeling using document context) can be identified as follows:

```
hasHeaderWord(S,X) :-
   word(W,X,_,_,_,_),
   hasWord(S,W),
   (atom(X) -> name(X,C) ; C = X),
   length(C,Len),
   Len > 4,
   all_upper(C).

signature isHeaderSentence(sent_id::
    sentence)::intensional.
isHeaderSentence(S) :-
   hasHeaderWord(S,_).

signature hasSectionHeader(sent_id::
    sentence, header::property)::
    intensional.
hasSectionHeader(S,X) :-
   nextS(S1,S),
   hasHeaderWord(S1,X).
hasSectionHeader(S,X) :-
   nextS(S1,S),
   not isHeaderSentence(S),
   once(hasSectionHeader(S1,X)).
```

In this case, first the sentences that contain a header word are identified using the helper pred-

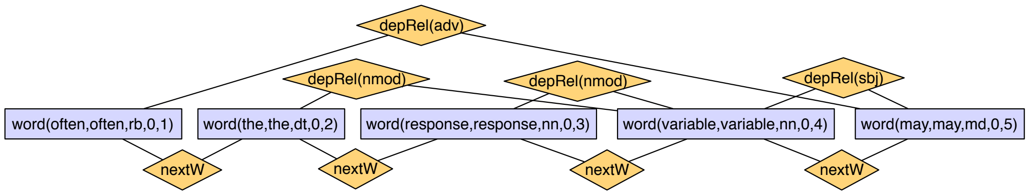

Figure 3: Graphicalization of the (partial) interpretation in Listing 3. For the sake of clarity, attributes of entities and relationships are depicted inside the respective entity or relationship.

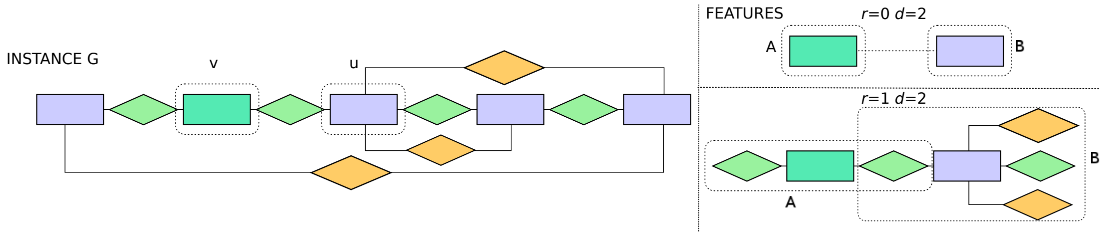

Figure 4: Illustration of the NSPDK feature concept. Left: instance G with 2 vertices v, u as roots for neighborhood subgraphs (A, B) at distance 2. Right: some of the neighborhood pairs, which form the NSPDK features, at distance $d = 2$ and radius $r = 0$ and 1 respectively. Note that neighborhood subgraphs can overlap.

icate hasHeaderWord, where a header word is defined as an upper case string that has more than four letters (lines 1-7). Next, all sentences that represent a section header are identified using the intensional signature isHeaderSentence (lines 9-11), and each sentence in the paragraphs following a particular section header is labeled with this header, using the hasSectionHeader predicate (lines 13-20).

Due to the relational approach, the span can be very large. Furthermore, since these features are defined declaratively, there is no need to reprocess the dataset each time a new feature is introduced, which renders experimentation very flexible[4].

3 Graphicalization and Feature Generation

In this step, a technique called *graphicalization* transforms the relational representations from the previous step into graph-based ones and derives features from a grounded entity/relationship diagram using graph kernels. This can be interpreted as unfolding the E/R diagram over the data. An example of the graphicalization of the interpretation part in Listing 3 can be found in Figure 3.

From the resulting graphs, features can be extracted using a feature generation technique that is based on Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) (Costa and De Grave, 2010), a particular type of graph kernel. Informally the idea of this kernel is to decompose a graph into small neighborhood subgraphs of increasing radii $r \leq r_{max}$. Then, all pairs of such subgraphs whose roots are at a distance not greater than $d \leq d_{max}$ are considered as individual features. The kernel notion is finally given as the fraction of features in common between two graphs.

Formally, the kernel is defined as:

$$\kappa_{r,d}(G,G') = \sum_{\substack{A,B\in R^{-1}_{r,d}(G)\\ A',B'\in R^{-1}_{r,d}(G')}} \mathbf{1}_{A\cong A'} \cdot \mathbf{1}_{B\cong B'} \qquad (1)$$

where $R^{-1}_{r,d}(G)$ indicates the multiset of all pairs of neighborhoods of radius r with roots at distance d that exist in G, and where $\mathbf{1}$ denotes the indicator function and $\cong$ the isomorphism between graphs. For the full details, we refer the reader to (Costa and De Grave, 2010). The neighborhood pairs are illustrated in Figure 4 for a distance of 2 between two arbitrary roots (v and u).

In kLog, the feature set is generated in a combinatorial fashion by explicitly enumerating all pairs of neighborhood subgraphs; this yields a high-dimensional feature space that is much richer than most of the direct propositionalization approaches. The result is an extended high-dimensional feature space on which a statistical learning algorithm can be applied. The feature generator is initialized using the new_feature_generator predicate and hyperparameters (e.g., maximum distance and radius, and match type) can be set using the kLog flags mechanism (Listing 2, lines 6-10).

[4] Note that changes in the *extensional* signatures do require reprocessing the dataset. However, for different runs of an experiment with varying parameters for the feature generator or the learner, kLogNLP uses a caching mechanism to check if the extensional signatures have changed, when calling the dataset predicate.

4 Learning

In the last step, different learning tasks can be performed on the resulting extended feature space. To this end, kLog interfaces with several solvers, including LibSVM (Chang and Lin, 2011) and SVM SGD (Bottou, 2010). Lines 11-15 (Listing 2) illustrate the initialization of LibSVM and its use for 10-fold cross-validation.

5 Conclusions and Future Work

In this paper, we presented kLogNLP, a natural language processing module for kLog. Based on an entity-relationship representation of the domain, it transforms a dataset into the graph-based relational format of kLog. The basic kLogNLP model can be easily extended with additional background knowledge by adding relations using the declarative programming language Prolog. This offers a more flexible way of experimentation, as new features can be constructed on top of existing ones without the need to reprocess the dataset. In future work, interfaces will be added to other (domain-specific) NLP frameworks (e.g., the BLLIP parser with the self-trained biomedical parsing model (McClosky, 2010)) and additional dataset formats will be supported.

Acknowledgments

This research is funded by the Research Foundation Flanders (FWO project G.0478.10 - Statistical Relational Learning of Natural Language). KDG was supported by ERC StG 240186 "MiGraNT".

References

Laura Antanas, Paolo Frasconi, Fabrizio Costa, Tinne Tuytelaars, and Luc De Raedt. 2012. A relational kernel-based framework for hierarchical image understanding. In *Lecture Notes in Computer Science,*, pages 171–180. Springer, November.

Laura Antanas, McElory Hoffmann, Paolo Frasconi, Tinne Tuytelaars, and Luc De Raedt. 2013. A relational kernel-based approach to scene classification. *IEEE Workshop on Applications of Computer Vision*, 0:133–139.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python.* O'Reilly Media, Inc., 1st edition.

Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proc. of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187. Springer.

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

Peter Pin-Shan Chen. 1976. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, March.

Fabrizio Costa and Kurt De Grave. 2010. Fast neighborhood subgraph pairwise distance kernel. In *Proc. of the 26th International Conference on Machine Learning,*, pages 255–262. Omnipress.

Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors. 2008. *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*. Springer.

Paolo Frasconi, Fabrizio Costa, Luc De Raedt, and Kurt De Grave. 2012. klog: A language for logical and relational learning with kernels. *CoRR*, abs/1205.3981.

Su Kim, David Martinez, Lawrence Cavedon, and Lars Yencken. 2011. Automatic classification of sentences to support evidence based medicine. *BMC Bioinformatics*, 12(Suppl 2):S5.

Parisa Kordjamshidi, Paolo Frasconi, Martijn van Otterlo, Marie-Francine Moens, and Luc De Raedt. 2012. Relational learning for spatial relation extraction from natural language. In *Inductive Logic Programming*, pages 204–220. Springer.

David McClosky. 2010. *Any Domain Parsing: Automatic Domain Adaptation for Natural Language Parsing*. Ph.D. thesis, Brown University, Providence, RI, USA. AAI3430199.

N. Rizzolo and D. Roth. 2010. Learning based java for rapid development of nlp systems. In *LREC*, Valletta, Malta, 5.

Mathias Verbeke, Paolo Frasconi, Vincent Van Asch, Roser Morante, Walter Daelemans, and Luc De Raedt. 2012a. Kernel-based logical and relational learning with kLog for hedge cue detection. In *Proc. of the 21st International Conference on Inductive Logic Programming*, pages 347–357. Springer, March.

Mathias Verbeke, Vincent Van Asch, Roser Morante, Paolo Frasconi, Walter Daelemans, and Luc De Raedt. 2012b. A statistical relational learning approach to identifying evidence based medicine categories. In *Proc. of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 579–589. ACL.