Journal of Machine Learning Research 18 (2018) 1-5 Submitted 2/16; Revised 10/17; Published 4/18

KeLP: a Kernel-based Learning Platform

Simone Filice filice@info.uniroma2.it
DICII, University of Roma, Tor Vergata, Italy
Giuseppe Castellucci castellucci@ing.uniroma2.it
DIE, University of Roma, Tor Vergata, Italy
Giovanni Da San Martino gmartino@hbku.edu.qa
Qatar Computing Research Institute, HKBU, Qatar
Alessandro Moschitti* amosch@amazon.com
Amazon
Danilo Croce croce@info.uniroma2.it
Roberto Basili basili@info.uniroma2.it
DII, University of Roma, Tor Vergata, Italy

Editor: Cheng Soon Ong

Abstract

KeLP is a Java framework that enables fast and easy implementation of kernel functions over discrete data, such as strings, trees or graphs and their combination with standard vectorial kernels. Additionally, it provides several kernel-based algorithms, e.g., online and batch kernel machines for classification, regression and clustering, and a Java environment for easy implementation of new algorithms. KeLP is a versatile toolkit, very appealing both to experts and practitioners of machine learning and Java language programming, who can find extensive documentation, tutorials and examples of increasing complexity on the accompanying website. Interestingly, KeLP can be also used without any knowledge of Java programming through command line tools and JSON/XML interfaces enabling the declaration and instantiation of articulated learning models using simple templates. Finally, the extensive use of modularity and interfaces in KeLP enables developers to easily extend it with their own kernels and algorithms.

Keywords: Kernel Machines, Structured Data and Kernels, Java Framework.

1. Introduction

Kernel methods for discrete structures (Shawe-Taylor and Cristianini, 2004) are popular and effective techniques for the design of learning algorithms on non-vectorial data, such as strings (Lodhi et al., 2002), trees (Collins and Duffy, 2002; Moschitti, 2006; Aiolli et al., 2009; Croce et al., 2011; Annesi et al., 2014) and graphs (Gärtner, 2003; Borgwardt and Kriegel, 2005; Shervashidze, 2011). These kernels are very valuable to model complex relations in real-world applications, where data naturally has a structured form, e.g., strings and graphs are used to represent DNA and chemical compounds, or parse trees can encode syntactic and semantic information expressed in text.

However, current software for structural kernels is mainly limited to specific research, and is often not made publicly available or easily adaptable to new application domains. SVM-Light-TK toolkit by Moschitti (2006) is one of few exceptions that provides the user with different string and tree kernels but no graph kernels. It is written in C language

*. Professor at the University of Trento, Italy.

thus extending it with new kernels can be costly, especially when new data structures are required. This may also prevent non programmers to use it for their specific applications.

In designing KeLP, we have capitalized on our previous experience with SVM-Light-TK and other toolkits to foster the reuse of previous software and models as well as their extendibility. We provide a software platform for learning on structured data, which is both easy to use for unexperienced users and easily extendable for developers. KeLP includes many standard kernel algorithms for classification, regression and clustering as well as popular kernel functions for strings, trees and graphs. Additionally, it includes kernel functions for modeling relations between pairs of objects, which are, e.g., required in paraphrasing detection, textual entailment and question answering (Moschitti and Zanzotto, 2007; Filice et al., 2015; Tymoshenko and Moschitti, 2015). Most importantly, new data structures, models, algorithms and kernels can be easily added on top of the previous code, facilitating and promoting the development of a library of kernel-based algorithms for structured data.

The KeLP source code is distributed under the terms of Apache 2.0 License. No additional software is required to be installed in order to use it, the Apache Maven project management tool resolves all module dependencies automatically. We also provide and maintain a website with updated tutorials and documentation.

2. The KeLP Framework: an Overview

KeLP is written in Java and uses three different Maven projects to logically separate its three main components: (*i*) the framework backbone implements classification, regression and clustering algorithms operating on vector-based kernels. These core modules along with SVMs[1] are always part of any framework instantiation. (*ii*) Additional-algorithm packages, e.g., online kernel machines, Nyström method (Williams and Seeger, 2001) and label sequence learning (Altun et al., 2003), and (*iii*) additional-kernel packages, which include kernel functions for sequences, trees and graphs. A complete and up-to-date list of algorithms and kernel functions, a full Javadoc API documentation in PDF, and tutorials for both end-users and developers are hosted on the KeLP website, `http://www.kelp-ml.org`.

2.1 Machine Learning Algorithms

Learning algorithms in KeLP are implemented following *implementation contracts* provided by specific Java interfaces for different scenarios, i.e., classification, regression and clustering, according to two main learning paradigms, i.e., batch and online. New learning algorithms can implement these interfaces, thus becoming fully integrated with the other library functions. More in detail: (*i*) The `ClassificationLearningAlgorithm` interface supports the definition of classification learning methods, such as SVMs (Chang and Lin, 2011) or the Dual Coordinate Descent (Hsieh et al., 2008). (*ii*) The `RegressionLearningAlgorithm` interface supports the definition of regressors, such as ϵ-SVR (Chang and Lin, 2011). (*iii*) The `ClusteringAlgorithm` interface enables the implementation of clustering algorithms, such as (Kulis et al., 2005). (*iv*) The `OnlineLearningAlgorithm` interface supports the definition of online learning algorithms, e.g., Passive Aggressive (Crammer et al., 2006), or the Soft Confidence Weighted (Wang et al., 2012) algorithms. Finally, (*v*) the `MetaLearningAlgorithm` interface enables the design of committees, such as the multi-classification schemas, e.g., One-VS-One and One-VS-All.

1. We include it because of its wide use.

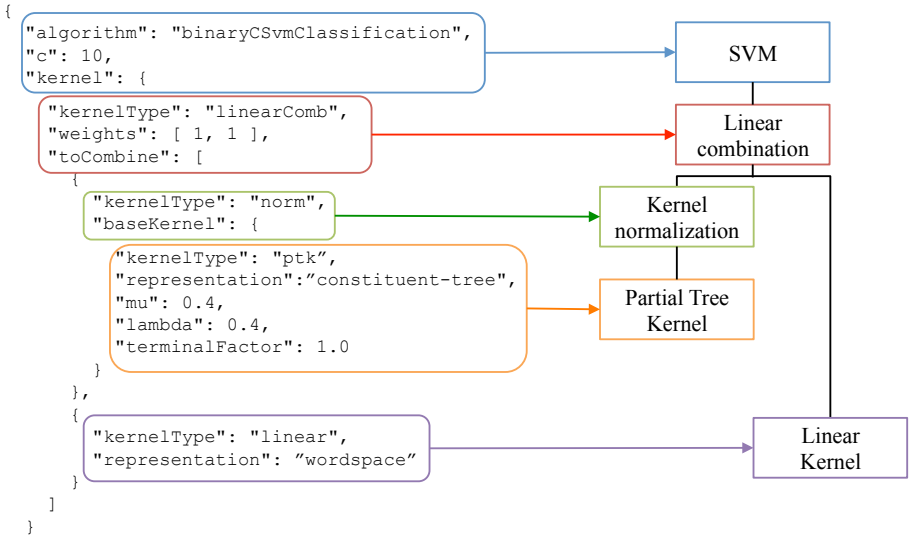

Figure 1: A JSON description of a SVM classifier.

2.2 Data Representation

In KeLP, data is represented by the `Example` class, which is constituted by (*i*) a set of `Labels` and (*ii*) a set of `Representations`. The former enables the design of single or multi-label classifiers and multi-variate regressors. The latter model examples in terms of vectors (e.g., `DenseVector` and `SparseVector`) or structures (e.g., `SequenceRepresentation`, `TreeRepresentation` or `GraphRepresentation`). In particular, kernels can be defined over examples encoded by multiple representations (e.g., multiple parse trees, strings, graphs and feature vectors). This makes the experimentation with multiple kernel combinations easy, just requiring negligible changes in the code or the JSON description (see Section 2.4), without the need of modifying the input data sets. Additionally, the examples can be combined in more complex structures, e.g., `ExamplePair`, useful to learn relations between objects, e.g., pairs representing question and answer text in QA, or text and hypothesis in textual entailment tasks. Building other types of data format is extremely simple, e.g., KeLP includes the SVM-Light-TK input format for trees and provides many scripts to use the popular gspan format for graphs (and indirectly for the 111 openBabel formats[2]).

2.3 Building Kernels from Kernels

KeLP enables (*i*) kernel composition, i.e., $K_{ab}(s_1, s_2) = (\phi_a \circ \phi_b)(s_1) \cdot (\phi_a \circ \phi_b)(s_2)$ from $K_a(s_1, s_2) = \phi_a(s_1) \cdot \phi_a(s_2)$ and $K_b(s_1, s_2) = \phi_b(s_1) \cdot \phi_b(s_2)$; and (*ii*) kernel combinations, e.g., $\lambda_1 K_a(s_1, s_2) + \lambda_2 K_b(s_1, s_2) \times K_a(s_1, s_2)$. These operations are coded using three abstractions of the `Kernel` class: (*i*) `DirectKernel` directly operates on a specified `Representation` object, derived from the `Example` object (e.g., implementing kernels for vectors, sequences, trees and graphs). (*ii*) The `KernelComposition` class composes `Kernel` objects, e.g., `PolynomialKernel`, `RBFKernel` and `NormalizationKernel`. (*iii*) `KernelCombination` class enables the combination of different `Kernels`, e.g., the `LinearKernelCombination` class applies a weighted kernel sum. (*iv*) `KernelOnPair` class operates

2. `http://openbabel.org`

on `ExamplePair`, e.g., to learn similarity functions between sentences (Filice et al., 2015) or to implement ranking algorithms with the `PreferenceKernel` class.

```
public static void run(String trainPath, String testPath, String learningAlgoPath){
        //Define (load) the learning algorithm  (see the JSON in Fig. 1)
        JacksonSerializerWrapper serializer = new JacksonSerializerWrapper();
        ClassificationLearningAlgorithm learningAlgo;
        learningAlgo = serializer.readValue(new File(learningAlgoPath),
                                        ClassificationLearningAlgorithm.class);
        //Load the datasets
        SimpleDataset trainDataset = new SimpleDataset();
        trainDataset.populate(trainPath);
        SimpleDataset testDataset = new SimpleDataset();
        testDataset.populate(testPath);
        //Learn the classifier
        List<Label> classes = trainDataset.getClassificationLabels();
        learningAlgo.setLabels(classes);
        learningAlgo.learn(trainDataset);
        //Classify and Evaluate
        Classifier classifier = learningAlgo.getPredictionFunction();
        Evaluator evaluator = new MulticlassClassificationEvaluator(classes);
        for(Example ex: testDataset.getExamples()){
                evaluator.addCount(ex, classifier.predict(ex));
        }
        System.out.println("ACC:" + evaluator.getPerformanceMeasure("accuracy"));
}
```

Listing 1: The Java instantiation (and evaluation) of the SVM classifier specified in Figure 1.

2.4 A User-friendly Interfacing with JSON

Each object, kernel function or algorithm, is serializable in JSON or XML. Thus, new algorithms can be implemented with a JSON description exploiting already implemented building blocks. The JSON interpreter of KeLP instantiates the corresponding objects without requiring any Java coding. Note that once a new kernel or learning algorithm is coded in Java, it will also be automatically available in the JSON format. Thus, it can be combined and composed with any kernel and algorithm available in KeLP by simply using JSON specifications. For example, Figure 1 provides a JSON description of an SVM classifier using a linear combination of a tree kernel with a linear kernel. The procedure for training and evaluating such classifier can be written in less than 20 code lines, as shown in Listing 1. Additionally, new kernels can be designed by combining JSON files and used in the framework by executing terminal commands (runnable jars). This enables experimenting with most KeLP features without writing any Java code.

3. Related Work and Conclusions

Most kernel-based software assumes data is represented by feature vectors (Hall et al., 2009; Chang and Lin, 2011; Abeel et al., 2009). Notable exceptions are SVM-Light-TK (Moschitti, 2006) and JKernelMachines (Picard et al., 2013). SVM-Light-TK is entirely written in C language and its main feature is the high computation speed. Unfortunately, C does not allow for fast prototyping of new kernel functions and machines. In contrast, KeLP enables fast and easy implementation of new kernel methods. JKernelMachines is a Java based package primarily designed to deal with custom kernels that cannot be easily found in standard libraries. However, many features offered by KeLP are not available in JKernelMachines, e.g., tree and graph kernels and regression algorithms. Moreover, KeLP supports easier composition and combination of kernels and learning algorithms.

References

Thomas Abeel, Yves Van de Peer, and Yvan Saeys. Java-ml: A machine learning library. *Journal of Machine Learning Research*, 10:931–934, 2009.

Fabio Aiolli, Giovanni Da San Martino, and Alessandro Sperduti. Route Kernels for Trees. In *Proceedings of ICML 09*, pages 17–24, Montreal, Quebec, Canada, 2009. ACM Press.

Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. Hidden markov support vector machines. In *Proceedings of ICML 2003*, pages 4–11, Menlo Park, CA, USA, August 2003.

Paolo Annesi, Danilo Croce, and Roberto Basili. Semantic compositionality in tree kernels. In *Proc. of CIKM 2014*, pages 1029–1038, New York, NY, USA, 2014. ACM.

Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-Path Kernels on Graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 74–81, 2005.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems*, pages 625–632. MIT Press, 2002.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *JMLR*, 7:551–585, December 2006.

Danilo Croce, Alessandro Moschitti, and Roberto Basili. Structured lexical similarity via convolution kernels on dependency trees. In *EMNLP*, pages 1034–1046, 2011.

Simone Filice, Giovanni Da San Martino, and Alessandro Moschitti. Structural representations for learning relations between pairs of texts. In *Proc. of ACL*, 2015.

Thomas Gärtner. A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1):49–58, 2003.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *sigkdd explor.*, 11(1), 2009.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proc. of ICML*, pages 408–415. ACM, 2008.

Brian Kulis, Sugato Basu, Inderjit Dhillon, and Raymond Mooney. Semi-supervised graph clustering: A kernel approach. In *Proc. of ICML*, pages 457–464. ACM, 2005.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, , and Chris Watkins. Text classification using string kernels. *journal of Machine Learning Research*, pages 419–444, 2002.

Alessandro Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of ECML'06*, pages 318–329, Berlin, Germany, 2006.

Alessandro Moschitti and Fabio Massimo Zanzotto. Fast and effective kernels for relational learning from texts. In *ICML'07*, pages 649–656. ACM, 2007.

David Picard, Nicolas Thome, and Matthieu Cord. Jkernelmachines: A simple framework for kernel machines. *Journal of Machine Learning Research*, 14:1417–1421, 2013.

John Shawe-Taylor and Nello Cristianini. *LaTeX User's Guide and Document Reference Manual*. Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.

Nino Shervashidze. Weisfeiler-lehman graph kernels. *The Journal of Machine Learning*, 12:2539–2561, 2011. URL `http://dl.acm.org/citation.cfm?id=2078187`.

Kateryna Tymoshenko and Alessandro Moschitti. Assessing the impact of syntactic and semantic structures for answer passages reranking. In *Proc. of CIKM*, pages 1451–1460. ACM, 2015.

Jialei Wang, Peilin Zhao, and Steven C. Hoi. Exact soft confidence-weighted learning. In John Langford and Joelle Pineau, editors, *Proceedings of ICML*, pages 121–128, 2012.

Christopher K. I. Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Proceedings of NIPS 2000*, 2001.