GraKeL: A Graph Kernel Library in Python

Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, Michalis Vazirgiannis

Journal of Machine Learning Research 21 (2020) 1-5 Submitted 6/18; Revised 3/20; Published 3/20

GraKeL: A Graph Kernel Library in Python

Giannis Siglidis YIANNIS.SIGLIDIS@LIP6.FR
LIP6, UPMC Université Paris 6, Sorbonne Universités
Paris, France
Giannis Nikolentzos NIKOLENTZOS@LIX.POLYTECHNIQUE.FR
Stratis Limnios STRATIS.LIMNIOS@POLYTECHNIQUE.EDU
Christos Giatsidis GIATSIDIS@LIX.POLYTECHNIQUE.FR
Konstantinos Skianis KSKIANIS@LIX.POLYTECHNIQUE.FR
LIX, École Polytechnique
Palaiseau, France
Michalis Vazirgiannis MVAZIRG@LIX.POLYTECHNIQUE.FR
LIX, École Polytechnique
Palaiseau, France
and
Department of Informatics, Athens University of Economics and Business
Athens, Greece

Editor: Antti Honkela

Abstract

The problem of accurately measuring the similarity between graphs is at the core of many applications in a variety of disciplines. Graph kernels have recently emerged as a promising approach to this problem. There are now many kernels, each focusing on different structural aspects of graphs. Here, we present GraKeL, a library that unifies several graph kernels into a common framework. The library is written in Python and adheres to the scikit-learn interface. It is simple to use and can be naturally combined with scikit-learn's modules to build a complete machine learning pipeline for tasks such as graph classification and clustering. The code is BSD licensed and is available at: `https://github.com/ysig/GraKeL`.

Keywords: graph similarity, graph kernels, scikit-learn, Python

1. Introduction

In recent years, graph-structured data has experienced an unprecedented growth in many domains, ranging from social networks to bioinformatics. Several problems of increasing interest involving graphs call for the use of machine learning techniques. Measuring the similarity or distance between graphs is a key component in many of those machine learning algorithms. Graph kernels have emerged as an effective tool for tackling the graph similarity problem. A graph kernel is a function that corresponds to an inner-product in a Hilbert space, and can be thought of as a similarity measure defined directly on graphs. The main advantage of graph kernels is that they allow a large family of machine learning algorithms, called kernel methods, to be applied directly to graphs.

GraKeL is a package that provides implementations of several graph kernels. The library is BSD licensed, and is publicly available on a GitHub repository encouraging collaborative work inside the machine learning community. The library is also compatible with scikit-learn, a standard package for performing machine learning tasks in Python (Pedregosa et al., 2011). Given scikit-learn's current inability to handle graph-structured data, the proposed library was built on top of one of its templates, and can serve as a useful tool for performing graph mining tasks. At the same time, it enjoys the overall object-oriented syntax and semantics defined by scikit-learn. Note that graphs are combinatorial structures and lack the convenient mathematical context of vector spaces. Hence, algorithms defined on graphs exhibit increased diversity compared to the ones defined on feature vectors. Therefore, bringing together all these kernels under a common framework is a challenging task, and the main design decisions behind GraKeL are presented in the following sections.

2. Underlying Technologies

Inside the Python ecosystem, there exist several packages that allow efficient numerical and scientific computation. GraKeL relies on the following technologies for implementing the currently supported graph kernels:

• NumPy: a package that offers all the necessary data structures for graph representation. Furthermore, it offers numerous linear algebra operations serving as a fundamental tool for achieving fast kernel calculation (Walt et al., 2011).

• SciPy: Python's main scientific library. It contains a large number of modules, ranging from optimization to signal processing. Of special interest to us is the support of sparse matrix representations and operations (Virtanen et al., 2020).

• Cython: allows the embedding of C code in Python. It is used to address efficiency issues related to non-compiled code in high-level interpreted languages such as Python, as well as for integrating low-level implementations (Behnel et al., 2011).

• scikit-learn: a machine learning library for Python. It forms the cornerstone of GraKeL since it provides the template for developing graph kernels. GraKeL can also interoperate with scikit-learn for performing machine learning tasks on graphs (Pedregosa et al., 2011).

• BLISS: a tool for computing automorphism groups and canonical labelings of graphs. It is used for checking graph isomorphism between small graphs (Junttila and Kaski, 2007).

• CVXOPT (optional): a package for convex optimization in Python. It is used for solving the semidefinite programming formulation that computes the Lovász number ϑ of a graph (Andersen et al., 2013).

3. Code Design

In GraKeL, all graph kernels are required to inherit the `Kernel` class which inherits from the scikit-learn's `TransformerMixin` class and implements the following four methods:

1. `fit`: Extracts kernel dependent features from an input graph collection.
2. `fit_transform`: Fits and calculates the kernel matrix of an input graph collection.
3. `transform`: Calculates the kernel matrix between a new collection of graphs and the one given as input to `fit`.
4. `diagonal`: Returns the self-kernel values of all the graphs given as input to `fit` along

with those given as input to `transform`, provided that this method has been called. This method is used for normalizing kernel matrices.

All kernels are unified under a submodule named `kernels`. They are all wrapped in a general class called `GraphKernel` which also inherits from scikit-learn's `TransformerMixin`. Besides providing a unified interface, it is also useful for applying other operations such as the the Nyström method, while it also facilitates the use of kernel frameworks that are currently supported by GraKeL. Frameworks like the Weisfeiler Lehman algorithm (Shervashidze et al., 2011) can use any instance of the `Kernel` class as their base kernel.

The input is required to be an `Iterable` collection of graph representations. Each graph can be either an `Iterable` consisting of a graph representation object (e. g., adjacency matrix, edge dictionary), vertex attributes and edge attributes or a `Graph` class instance. The vertex and edge attributes can be discrete (a.k.a. vertex and edge labels in the literature of graph kernels) or continuous-valued feature vectors. Note that some kernels cannot handle vector attributes, while others assume unlabeled graphs. Furthermore, through its `datasets` submodule, GraKeL facilitates the application of graph kernels to several popular graph classification datasets contained in a public repository (Kersting et al., 2016).

4. Comparison to Other Software

In the past years, researchers in the field of graph kernels have made available small collections of graph kernels. These kernels are written in various languages such as Matlab and Python, and do not share a general common structure that would provide an ease for usability. In the absence of software packages to compute graph kernels, the `graphkernels` library was recently developed (Sugiyama et al., 2017). All kernels are implemented in C++, while the library provides wrappers to R and Python. The above packages and the `graphkernels` library exhibit limited flexibility since kernels are not wrapped in a meaningful manner and their implementation does not follow object-oriented concepts. GraKeL, on the other hand, is a library that employs object-oriented design principles encouraging researchers and developers to integrate their own kernels into it.

Moreover, the `graphkernels` library contains only a handful of kernels, while several state-of-the-art kernels are missing. On the other hand, GraKeL provides implementations of a larger number of kernels. In a quick comparison, the `graphkernels` library provides variations of 5 kernels and 1 kernel framework, while GraKeL provides implementations of 15 kernels and 2 kernel frameworks. Moreover, GraKeL is compatible with the scikit-learn pipeline allowing easy and fast integration inside machine learning algorithms. In addition, given the diversities in the evaluation of machine learning methods, GraKeL provides a common ground for comparing existing kernels against newly designed ones. This can be of great interest to researchers trying to evaluate kernels they have come up with. It should also be mentioned that GraKeL is accompanied by detailed documentation including several examples of how to apply graph kernels to real-world data.

Furthermore, even though GraKeL is implemented in Python, as shown in Figure 1 below, several of its kernels are more efficient than the corresponding implementations in `graphkernels`. Due to space limitations, we only present the results for a single benchmark dataset (i. e. ENZYMES). The rest of the results can be found in the documentation[1].

1. `https://ysig.github.io/GraKeL/latest/benchmarks/comparison.html`

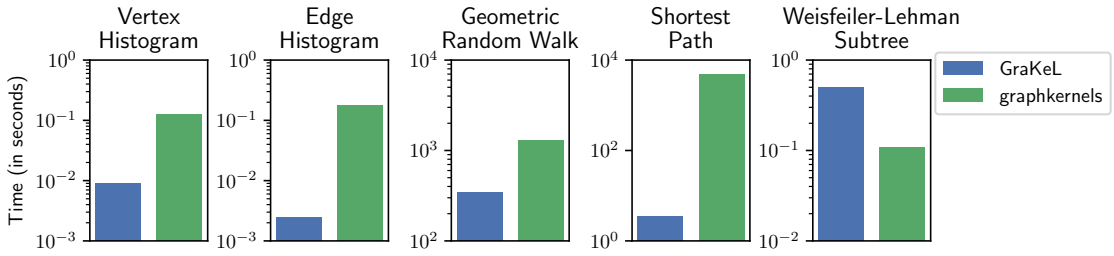

Figure 1: Running time (in seconds) for kernel computation on the ENZYMES dataset using the GraKeL and `graphkernels` libraries.

5. Sample Code

The most common use of a graph kernel is the one where given a collection of training graphs $\mathcal{G}_n$ (of size n) and a collection of test graphs $\mathcal{G}_m$ (of size m), the goal is to compute two separate kernel matrices: (1) an $n \times n$ matrix between all the graphs of $\mathcal{G}_n$, and (2) a $m \times n$ matrix between the graphs of $\mathcal{G}_m$ and those of $\mathcal{G}_n$. This can be accomplished by running the `fit_transform` method on $\mathcal{G}_n$, and then the `transform` method on $\mathcal{G}_m$. Then, these matrices can be passed on to the SVM classifier to perform graph classification. The following example demonstrates the use of GraKeL for performing graph classification on a standard dataset.

```
>>> from grakel.datasets import fetch_dataset
>>> from sklearn.model_selection import train_test_split
>>> from grakel.kernels import ShortestPath
>>> from sklearn.svm import SVC
>>> from sklearn.metrics import accuracy_score
>>>
>>> MUTAG = fetch_dataset("MUTAG", verbose=False)
>>> G, y = MUTAG.data, MUTAG.target
>>> G_train, G_test, y_train, y_test = train_test_split(G, y, test_size=0.1,
                                                        random_state=42)
>>>
>>> sp_kernel = ShortestPath()
>>> K_train = sp_kernel.fit_transform(G_train)
>>> K_test = sp_kernel.transform(G_test)
>>>
>>> clf = SVC(kernel='precomputed').fit(K_train, y_train)
>>> y_pred = clf.predict(K_test)
>>> print("accuracy: %2.2f %%" %(accuracy_score(y_test, y_pred)*100))
accuracy: 84.21 %
```

6. Conclusion

GraKeL is a library that implements several state-of-the-art graph kernels, while remaining user-friendly. It relies on the scikit-learn's pipeline, and it can thus be easily integrated into various machine learning applications.

Acknowledgments

We would like to thank the editor and the anonymous reviewers for their constructive comments. This work was supported by the Labex DigiCosme "Grakel" project.

References

Martin S Andersen, Joachim Dahl, and Lieven Vandenberghe. CVXOPT: A Python package for convex optimization. *Available at cvxopt. org*, 2013.

Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The Best of Both Worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

Tommi Junttila and Petteri Kaski. Engineering an Efficient Canonical Labeling Toolfor Large and Sparse Graphs. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments and the 4th Workshop on Analytic Algorithms and Combinatorics*, pages 135–149, 2007.

Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark Data Sets for Graph Kernels, 2016. URL `http://graphkernels.cs.tu-dortmund.de`.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, and David Cournapeau. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

Mahito Sugiyama, M Elisabetta Ghisu, Felipe Llinares-López, and Karsten Borgwardt. graphkernels: R and Python packages for graph comparison. *Bioinformatics*, 34(3):530–532, 2017.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, pages 1–12, 2020.

Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.