Journal of Machine Learning Research 1 (2018) 1-48 Submitted 4/00; Published 10/00

GraKeL: A Graph Kernel Library in Python

Giannis Siglidis IOANNIS.SIGLIDIS@POLYTECHNIQUE.EDU
Giannis Nikolentzos NIKOLENTZOS@LIX.POLYTECHNIQUE.FR
Stratis Limnios LIMNIOS.STRATIS@POLYTECHNIQUE.EDU
Christos Giatsidis GIATSIDIS@LIX.POLYTECHNIQUE.FR
Konstantinos Skianis KSKIANIS@LIX.POLYTECHNIQUE.FR
Michalis Vazirgianis MVAZIRG@LIX.POLYTECHNIQUE.FR
LIX, École Polytechnique, France

Editor:

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 is build on top of scikit-learn. 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

Python is an open-source, general-purpose programming language. Due to a number of attractive features, Python has established itself as one of the most popular programming languages [1]. Among others, Python offers a high-level programming interface along with a large number of scientific libraries (e. g., SciPy). It is thus not surprising why it has been widely adopted both in academia and in the industry.

In recent years, graph-structured data has experienced an enormous growth in many domains, ranging from social networks to bioinformatics. Several problems of increasing interest call for the use of machine learning techniques on such data. 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.

Currently, there are many graph kernels available, and each of them focuses on different structural aspects of graphs. GraKeL was created to provide state-of-the-art implementations of several of these kernels. Scikit-learn is a standard tool for performing machine

1. `https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages`

learning tasks in Python (Pedregosa et al., 2011). It provides a user-friendly interface, high-quality implementations and readable documentation written for a broad audience. Given its current inability to handle graph-structured data, the proposed library was built on top of one of its templates, and can thus 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. Due to the combinatorial nature of graphs, bringing together all these kernels under a common framework is a challenging task, and the main design decisions behind GraKeL are presented in detail.

2. What's New

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, both the `graphkernels` library and the collections of kernels mentioned above contain 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 allowing users to gain access to a wider range of algorithms. In a quick comparison, the `graphkernels` library provides variations of 3 kernels and a kernel framework, while GraKeL provides implementations of 15 kernels and 3 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. The library is BSD licensed, and is publicly available on a GitHub repository encouraging collaborative work inside the machine learning and data mining communities. Finally, it is `pip` installable which makes it a member of the Python ecosystem, having pre-built wheels in all platforms (Linux, OSX, Windows) and supporting various architectures (32/64bit for Linux and OSX, 64bit for Windows).

3. 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.
• 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.
• 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.
• scikit-learn: a machine learning library for Python. It forms the cornerstone of GraKeL since it provides the template for developing graph kernels. GraKeL is designed to interoperate with scikit-learn for performing machine learning tasks on graph-structured data.
• BLISS: a tool for computing automorphism groups and canonical labelings of graphs. It is written in C++ and is used for checking graph isomorphism between small graphs.
• 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.

4. Code Design

In GraKeL, all graph kernels are required to inherit the `Kernel` class which is something between a class and an interface inheritance. This class corresponds to the scikit-learn's `TransformerMixin` class and implements the following four methods for external use:

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 not included into the standard `TransformerMixin` class and is used for normalizing kernel matrices.

The `TransformerMixin` class also implements the following three internal methods that are usually overridden when implementing a kernel:

1. `parse_input`: Parses the input graph collection and calculates features for all the graphs contained in it.

2. `_calculate_kernel_matrix`: Computes the kernel matrix between two collections of graphs. Utilizes the features generated by `parse_input`.

3. `pairwise_operation`: If the above method is not overridden and `parse_input` provides an `Iterable` containing the features of each graph, then this method computes the kernel between each pair of those graphs.

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),

vertex attributes and edge attributes or a `Graph` class instance (as described below). 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.

All kernels are unified under a submodule named `kernels`. They are all decorated by a central class called `GraphKernel` which also corresponds to a scikit-learn pipeline-compatible `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 basic 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.

GraKeL represents each graph as an instance of the `Graph` class. The class supports various formats for graph-structured data (e. g., adjacency matrix, edge dictionary) and features a mechanism that automatically detects the format of an input graph. It also provides the developer with the ability to impose and to ignore the internal graph representation when using graph features inside a kernel calculation, while allowing the user to experience a uniform input profile for all kernels. The efficiency both in terms of memory consumption and in terms of application consistency that such a class offers, led us to this decision instead of employing a standard graph library.

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). Specifically, the `fetch_dataset` function downloads the dataset, transforms it into the format required by the `Graph` class, and then, it can be given as input to any `Kernel` object.

It is necessary to mention that the decision to follow the scikit-learn's template design may result in counterintuitive behavior regarding the runtime of the `fit` and `transform` methods. The most common use of a graph kernel is the one where given two collections of graphs $\mathcal{G}_n$ and $\mathcal{G}_m$ (of sizes n and m respectively), 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 $\mathcal{G}_m$. In some cases, the features extracted by the `fit` method form an object which is mutable when `transform` takes place. The additional copying cost for protecting that object will lead to the increase of the runtime of the whole process. Therefore, in such cases, it may be more efficient to merge the two collections, apply `fit_transform` to the new collection and extract the desired matrices from the emerging matrix instead of applying the two-step approach described above.

5. Conclusion

GraKeL implements a wide variety of state-of-the-art graph kernels, while being implemented on a user friendly interface. It relies on the scikit-learn's pipeline, and it can thus be easily integrated into various machine learning applications. Since most existing graph kernels follow either the R-convolution or the assignment kernel design paradigm, we aim at extending GraKeL in a consistent (object oriented) manner to facilitate the integration of kernels belonging to these families.

Acknowledgments

This work is supported by the Labex DigiCosme "Grakel" project.

References

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, et al. 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.