Supplementary Material for graphkernels: 
R and Python packages for graph comparison

Mahito Sugiyama  
National Institute of Informatics;  
JST PRESTO  
mahito@nii.ac.jp

M. Elisabetta Ghisu  
D-BSSE, ETH Zürich;  
Swiss Institute of Bioinformatics  
elisabetta.ghisu@bsse.ethz.ch

Felipe Llinares-López  
D-BSSE, ETH Zürich;  
Swiss Institute of Bioinformatics  
felipe.llinares@bsse.ethz.ch

Karsten M. Borgwardt  
D-BSSE, ETH Zürich;  
Swiss Institute of Bioinformatics  
karsten.borgwardt@bsse.ethz.ch

1 Definitions of Graph Kernels

Let $G = (V, E, \varphi)$ be a labeled graph, where $V$ is the vertex set, $E$ is the edge set, and $\varphi$ is a mapping $\varphi: V \cup E \to \Sigma$ with the range $\Sigma = \{1, 2, \ldots, s\}$ of vertex and edge labels. For an edge $(u, v) \in E$, we identify $(u, v)$ and $(v, u)$ if $G$ is undirected. In the following, we assume that the range of labels $\Sigma = \{1, 2, \ldots, s\}$ without loss of generality. Given a collection of $n$ graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$ as a dataset with $G_i = (V_i, E_i, \varphi_i)$ for each $i \in \{1, 2, \ldots, n\}$, each of the following functions in our package graphkernels returns the kernel (Gram) matrix $(K_{ij}) \in \mathbb{R}^{n \times n}$ with the respective graph kernel, where each $K_{ij} = k_{\text{kernel-type}}(G_i, G_j)$.

- **CalculateVertexHistKernel($\mathcal{G}$)** calculates the kernel matrix of the linear kernel $k_{\text{VH}}$ between vertex label histograms given as
  \[
  K_{ij} = k_{\text{VH}}(G_i, G_j) = \langle f_i, f_j \rangle = \sum_{l=1}^{s} f^{l}_i f^{l}_j,
  \]
  where each vertex label histogram $f_j$ is a vector $(f^{1}_i, f^{2}_i, \ldots, f^{s}_i)$, such that $f^{l}_i = |\{v \in V_i \mid \varphi_i(v) = l\}|$ for each $l \in \Sigma$.

- **CalculateEdgeHistKernel($\mathcal{G}$)** calculates the kernel matrix of the linear kernel $k_{\text{EH}}$ between edge label histograms given as
  \[
  K_{ij} = k_{\text{EH}}(G_i, G_j) = \langle g_i, g_j \rangle = \sum_{l=1}^{s} g^{l}_i g^{l}_j.
  \]
  where the edge label histogram $g_j = (g^{1}_j, g^{2}_j, \ldots, g^{s}_j)$ such that $g^{l}_j = |\{(u, v) \in E_j \mid \varphi_j(u, v) = l\}|$ for each $l \in \Sigma$.

- **CalculateVertexEdgeHistKernel($\mathcal{G}$)** calculates the kernel matrix of the linear kernel $k_{\text{VEH}}$ between vertex-edge label histograms given as
  \[
  K_{ij} = k_{\text{VEH}}(G_i, G_j) = \langle h_i, h_j \rangle = \sum_{l_1, l_2, l_3=1}^{s} h^{l_1} i^{l_2} j^{l_3}.
  \]
where the vertex-edge label histogram \( h_i = (h_i^{111}, h_i^{211}, \ldots, h_i^{313}) \), such that \( h_i^{l_1,l_2,l_3} = |\{(u,v) \in E_i \mid \varphi_i(u,v) = l_1, \varphi_i(u) = l_2, \text{ and } \varphi_i(v) = l_3\}| \) for each \( l_1, l_2, l_3 \in \Sigma \). Notice that \( k_{\text{VEH}}(G,G') = k_{\text{EH}}(G,G') \) if vertices are not labeled.

- **CalculateVertexVertexEdgeHistKernel** \((G, \lambda)\) calculates the kernel matrix of the following histogram kernel \( k_H \):
  \[
  K_{ij} = k_H(G_i, G_j) = k_{\text{VH}}(G_i, G_j) + \lambda k_{\text{VEH}}(G_i, G_j).
  \]

- **CalculateVertexHistGaussKernel** \((G, \sigma)\) calculates the kernel matrix of the Gaussian RBF kernel \( k_{\text{VH,G}} \) between vertex label histograms given as
  \[
  K_{ij} = k_{\text{VH,G}}(G_i, G_j) = \exp\left(-\frac{\|f_i - f_j\|^2}{2\sigma^2}\right).
  \]

- **CalculateEdgeHistGaussKernel** \((G, \sigma)\) calculates the kernel matrix of the Gaussian RBF kernel \( k_{\text{EH,G}} \) between edge label histograms given as
  \[
  K_{ij} = k_{\text{EH,G}}(G_i, G_j) = \exp\left(-\frac{\|g_i - g_j\|^2}{2\sigma^2}\right).
  \]

- **CalculateVertexEdgeHistGaussKernel** \((G, \sigma)\) calculates the kernel matrix of the Gaussian RBF kernel \( k_{\text{VEH,G}} \) between vertex-edge label histograms given as
  \[
  K_{ij} = k_{\text{VEH,G}}(G_i, G_j) = \exp\left(-\frac{\|h_i - h_j\|^2}{2\sigma^2}\right).
  \]

- **CalculateGraphletKernel** \((G, k)\) calculates the kernel matrix of the graphlet kernel \( k_{\text{GL}} \) given as
  \[
  K_{ij} = k_{\text{GL}}(G_i, G_j) = \langle l_i, l_j \rangle,
  \]
where \( l_i \) is a vector \((l_i^1, l_i^2, \ldots, l_i^k)\) such that each \( l_i^j \) is the number of embeddings of the \( j \)th graphlet \( L_j \) in \( G_i \). Each graphlet \( L_j \) is a graph with \( k \) nodes, and \( L_1, L_2, \ldots, L_k \) is the list of all connected and disconnected graphlets. Mathematically, a graphlet \( L = (V_L, E_L) \) is embedded in \( G \) if there is an injective mapping \( \alpha : V_L \rightarrow V \) such that \((v,w) \in E_L\) if and only if \((\alpha(v), \alpha(w)) \in E\). Due to the high computational cost, this function supports only \( k \in \{3,4\} \), and it ignores vertex and edge labels. See (Shervashidze et al., 2009) for the detailed information.

- **CalculateConnectedGraphletKernel** \((G, k)\) calculates the kernel matrix of the graphlet kernel \( k_{\text{GL}} \) with connected graphlets given as
  \[
  K_{ij} = k_{\text{CGL}}(G_i, G_j) = \langle l_i, l_j \rangle,
  \]
where \( l_i \) is a vector \((l_i^1, l_i^2, \ldots, l_i^k)\) such that each \( l_i^j \) is the number of embeddings of the \( j \)th graphlet \( L_j \) in \( G_i \) and only connected graphlets are considered in this kernel. This function supports \( k \in \{3,4,5\} \), and it ignores vertex and edge labels.
• CalculateKStepRandomWalkKernel\((G, (\lambda_0, \lambda_1, \ldots, \lambda_\kappa))\) calculates the kernel matrix of the \(\kappa\)-step random walk kernel \(k_{\kappa}^\chi\) given as

\[
K_{ij} = k_{\kappa}^\chi(G_i, G_j) = \sum_{t', j'=1}^{\vert V_i \vert} \left[ \sum_{t=0}^{\kappa} \lambda_t A_t^i \right]_{t', j'},
\]

where \(A_\chi\) is the adjacency matrix of the direct (tensor) product \(G_\chi = (V_\chi, E_\chi, \varphi_\chi)\) between \(G_i = (V_i, E_i, \varphi_i)\) and \(G_j = (V_j, E_j, \varphi_j)\) (Borgwardt, 2007; Gärtner et al., 2003; Vishwanathan et al., 2010) such that

\[
V_\chi = \{ (v_i, v_j) \in V_i \times V_j \mid \varphi_i(v_i) = \varphi_j(v_j) \},
\]

\[
E_\chi = \{ ((u_i, u_j), (v_i, v_j)) \in V_\chi \times V_\chi \mid (u_i, v_i) \in E_i, (u_j, v_j) \in E_j, \text{ and } \varphi_i(u_i, v_i) = \varphi_j(u_j, v_j) \},
\]

and all labels are inherited, that is,

\[
\varphi_\chi((v_i, v_j)) = \varphi_i(v_i) = \varphi_j(v_j),
\]

\[
\varphi_\chi((u_i, u_j), (v_i, v_j)) = \varphi_j(u_i, v_i) = \varphi_j(u_j, v_j).
\]

Each weight \(\lambda_t\) for \(t \in \{0, 1, \ldots, \kappa\}\) should be positive real-valued.

• CalculateGeometricRandomWalkKernel\((G, \lambda)\) calculates the kernel matrix of the geometric random walk kernel \(k_{GR}\) given as

\[
K_{ij} = k_{GR}(G_i, G_j) = \sum_{t', j'=1}^{\vert V_i \vert} \left[ \sum_{t=0}^{\infty} \lambda^t A_t^i \right]_{t', j'} = \sum_{t', j'=1}^{\vert V_i \vert} \left[ (I - \lambda A_\chi)^{-1} \right]_{t', j'},
\]

where \(A_\chi\) is the adjacency matrix of the direct product of \(G_i\) and \(G_j\). Note that \(\lambda < 1/\mu_{\chi,\text{max}}\) should be satisfied with the maximum eigenvalue \(\mu_{\chi,\text{max}}\) of \(A_\chi\). Among the random walk based kernels (Borgwardt et al., 2005; Gärtner et al., 2003; Kashima et al., 2003; Mahé et al., 2004), the geometric random walk kernel is often used as a baseline, while Sugiyama and Borgwardt (2015) pointed out that the \(\kappa\)-step random walk kernel is more appropriate for the baseline than the geometric random walk kernel.

• CalculateExponentialRandomWalkKernel\((G, \beta)\) calculates the kernel matrix of the exponential random walk kernel \(k_{ER}\) by setting each \(\lambda_t = \beta^t / t!\) given as

\[
K_{ij} = k_{ER}(G_i, G_j) = \sum_{t', j'=1}^{\vert V_i \vert} \left[ \sum_{t=0}^{\infty} \frac{(\beta A_\chi)^t}{t!} \right]_{t', j'} = \sum_{t', j'=1}^{\vert V_i \vert} \left[ e^{\beta A_\chi} \right]_{t', j'},
\]

where \(A_\chi\) is the adjacency matrix of the direct product of \(G_i\) and \(G_j\).

• CalculateShortestPathKernel\((G)\) calculates the kernel matrix of the shortest-path kernel \(k_{SP}\) given as

\[
K_{ij} = k_{SP}(G_i, G_j) = k_{1}^\chi(F(G_i), F(G_j)),
\]

where \(\lambda_0 = 0\) and \(\lambda_1 = 1\) in the 1-step random walk kernel \(k_1^\chi\), and \(F(G) = (V_F, E_F, \varphi_F)\) is the Floyd transformed graph of \(G\) such that \(V_F = V, E_F = V \times V\) (i.e., the complete graph), and \(\varphi((u, v))\) is the length of the shortest path between \(u\) and \(v\) in \(G\). See (Borgwardt and Kriegel, 2005) for the detailed information.
CalculateWLKernel(G, h) calculates the Weisfeiler-Lehman subtree kernel \( k_{WL} \). For each graph \( G \in \mathcal{G} \), let \( G^{(i)} \) be the graph \((V, E, \varphi^{(i)})\) such that \( \varphi^{(1)}(v) = \varphi(v) \) and \( \varphi^{(i)} \) with \( i > 1 \) is defined as

\[
\varphi^{(i)}(v) = \left( \varphi^{(i-1)}(v), \{ \varphi^{(i-1)}(u) \mid (v, u) \in E \} \right)
\]

for each \( v \in V \), and let \( f^{(12...h)} \) be the concatenation of vertex label histograms for \( G^{(1)}, G^{(2)}, \ldots, G^{(h)} \). The Weisfeiler-Lehman subtree kernel is the linear kernel between concatenated vertex label histograms, that is,

\[
K_{ij} = k_{WL}(G_i, G_j) = \left\langle f^{(12...h)}_i, f^{(12...h)}_j \right\rangle.
\]

See (Shervashidze et al., 2011) for the detailed information.

2 Evaluation and Choice of Graph Kernels

As in (Shervashidze et al., 2011) and (Sugiyama and Borgwardt, 2015) we used the accuracy as a measure of performance, which is defined as the proportion of correctly classified data points over the entire sample.

A comprehensive overview of different graph kernels, including a detailed performance comparison on a variety of datasets is provided in (Shervashidze et al., 2011); as it can be observed, there is no kernel which is consistently superior, but the performance is rather dataset-dependent. Overall, the WL kernels is a competitive choice in terms of both running time and accuracy on all common benchmark datasets. For walk-based graph kernels, Sugiyama and Borgwardt (2015) provide guidelines on how to choose the best kernel among these.

In practice, one should choose a graph kernel based on the type of network feature (e.g. walks, shortest paths, subtree patterns) that is considered most informative in a given application. If no such prior knowledge is available, the kernel should be chosen by cross-validation on the training dataset.

3 Experimental Methodology

The MUTAG dataset used in our application contains 188 aromatic and heteroaromatic nitro compounds, and each compound is modeled as a graph. The objective is to predict labels of graphs, indicating whether or not they are a mutagenic compound as tested by the Ames test using Salmonella typhimurium. All experiments were run using R version 3.3.3 for macOS 10.12.4 on a single core of 4.0 GHz Intel Core i7 CPU and 32 GB of memory.

To reproduce Figure 2, first install R and the graphkernels library following instructions in Section 4. Since this experiment uses additional R packages kernlab, ggplot2, scales, and gridExtra, install them by running in R:

\[
> \text{install.packages("kernlab")}
> \text{install.packages("ggplot2")}
> \text{install.packages("scales")}
> \text{install.packages("gridExtra")}
\]

Then download the script file “script.R”, and run the following in the R environment.
4 Installation and Usage Instructions in R

The following is a step-by-step installation and usage guide of the graphkernels library in R. If R is already installed in your computer, please skip Section 4.1.

4.1 Install R and RStudio

1. Download a binary distribution of the R software at CRAN (https://cran.r-project.org/) and install it in your computer.
2. We recommend to use RStudio, which is an integrated development environment (IDE) for R and includes a console, a code editor, and visualization tools. You can install RStudio from a binary distribution available at https://www.rstudio.com/products/RStudio/.

4.2 Install graphkernels in R

1. To install the graphkernels library, launch R (or RStudio) and type at the console:

   > install.packages("graphkernels", dependencies = TRUE)

   The graphkernels library as well as all dependent libraries will be automatically installed.

4.3 How to Use graphkernels in R

1. Load the library in your R console by

   > library(graphkernels)

   Then you can use any functions in our library to compute a kernel matrix from a graph dataset.

2. Prepare a graph dataset. We assume that each graph is an igraph graph, and a graph dataset is a list of igraph graphs. Please see the igraph manual (https://cran.r-project.org/web/packages/igraph/igraph.pdf) for the details of the igraph library (Csardi and Nepusz, 2006). As an example of a graph dataset, our library contains the MUTAG dataset, which is loaded by

   > data(mutag)

   Each graph can be displayed as

   > mutag[[1]]
   IGRAPH f2f3caf U--- 23 27 --
   + attr: label (g/n), label (v/n), label (e/n)
   + edges from f2f3caf:
   [1] 1-- 2 1--14 2-- 3 3-- 4 3--12 4-- 5 5-- 6 6-- 7 7--21 8-- 9 9--10 10--11 10--16 11--12 12--13 13--14 5
We assume that vertex and edge labels are stored in the form of attributes of vertices and edges with the name “label”, which can be checked as

```r
> vertex_attr(mutag[[1]])$label
[1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 6 7 7
> edge_attr(mutag[[1]])$label
```

3. To compute a kernel matrix, use one of the kernel functions of the `graphkernels` library. For example, to compute the vertex label histogram kernel $k_{VH}$, type the following command:

```r
> K <- CalculateVertexHistKernel(mutag)
```

Then the kernel matrix of the linear kernel $k_{VH}$ between vertex label histograms is computed and stored in `K`. For example, the kernel value between graphs 1 and 2 is

```r
> K[1, 2]
[1] 282
```

If a kernel includes a parameter, it can be specified as an additional argument of a function. For example,

```r
> K <- CalculateWLKernel(mutag, 5)
```

Then the Weisfeiler-Lehman subtree kernel $k_{WL}$ with $h = 5$ is computed. In the $\kappa$-step random walk kernel $k_{\kappa}^\kappa$, each weight $\lambda_t$, $t = 0, 1, \ldots, \kappa$ should be provided as a vector, for instance,

```r
> K <- CalculateKStepRandomWalkKernel(mutag, c(1.0, 1.0, 1.0))
```

for $\kappa = 2$ and $\lambda_0 = \lambda_1 = \lambda_2 = 1$. See Section 1 for the entire list of graph kernels supported by the `graphkernels` library. In R, you can directly see the manual of each graph kernel by the command “?<the_name_of_function>”, for example,

```r
> ?CalculateVertexHistKernel
```

which includes short summary of the kernel, parameters (arguments), and example usages.

5 Installation and Usage Instructions in Python

We provide below some information on the installation and usage of our Python package. We recommend the users to also refer to our GitHub page (https://github.com/eghisu/GraphKernels), where the latest updates will be provided. If Python is already installed, please skip Section 5.1.

5.1 Install Python

We recommend the users to follow the instructions at https://www.python.org/downloads/ for the installation of Python.
By default, Python should be already installed on Ubuntu and macOS machines. However, we still recommend the users to refer to the page above, in order to be up-to-date with the latest releases.

5.2 Install graphkernels in Python

We recommend the users to install the package via pip, by typing in a terminal

g pip install graphkernels

This command will also automatically install all the dependent libraries.

Alternatively, the package can be installed from source. After downloading the source code from our GitHub repository (https://github.com/eghisu/GraphKernels/tree/master/graphkernels) or from pypi (https://pypi.python.org/pypi/graphkernels), the users can use the setup.py script to install the package, by typing

g python setup.py install

5.3 How to use graphkernels in Python

We illustrate an example usage of our Python package using the benchmark dataset MUTAG (Debnath et al., 1991), which is also provided with our package. We also provide an online tutorial on our GitHub (https://github.com/eghisu/GraphKernels/tree/master/graphkernelsTutorial).

1. Load the required packages. Import the graphkernels library and numpy library.

   ```
   >>> import graphkernels.kernels as gk
   >>> import numpy as np
   ```

2. Load the data.

   ```
   >>> # Load the data in the graphkernels package folder
   >>> data = np.load("graphkernels/data.mutag")
   ```

   Please, note that the path above might need to be updated, depending on the directory where the data.mutag file is saved. In particular, the line above should look like:

   ```
   >>> data = np.load("dataDir/data.mutag"),
   ```

   where dataDir is the path to the data.mutag file.

3. Compute the kernel matrix. We provide below a few example of kernels which can be computed with our package.

   ```
   >>> # Vertex Histogram Kernels: compute and store the kernel matrix K
   >>> K = gk.CalculateVertexHistKernel(data)
   >>> K[0,1]
   282.0
   ```
The matrix $K$ above stores the kernel matrix of the linear kernel $k_{VH}$ between vertex label histograms. If a kernel includes a parameter, or if the user would like to change the default parameter value, this can be provided as an extra argument. For instance,

```python
>>> K = gk.CalculateWLKernel(data, 5)
```

computes the WL kernel $k_{WL}$ for the parameter $h = 5$.

Another example is the random walk kernel, where a list of $\lambda$ parameters can be provided.

```python
>>> # 5-step random walk kernel: compute and store the kernel matrix K
>>> K = gk.CalculateKStepRandomWalkKernel(data, [1,1,1,1,1])
>>> K[0,1]
398942.0
```

In particular, for the KStepRandomWalkKernel example above, $k = 4$ and $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1$.

References


