# Homework 6: Random Walk Kernel and DBSCAN

Christian Bock
christian.bock@bsse.ethz.ch

Dr. Felipe Llinares
felipe.llinares@bsse.ethz.ch

Dr. Bastian Rieck
bastian.rieck@bsse.ethz.ch

Prof. Dr. Karsten Borgwardt
karsten.borgwardt@bsse.ethz.ch

*Submission deadline: 19.12.2018 at 08:00*

## Objectives

The goals of this homework are:

- to understand and implement the random walk graph kernel algorithm.
- to understand its computational complexity.
- to understand the principle of the DBSCAN clustering algorithm.

## Problem overview

In this homework you will investigate the random walk kernel [Gärtner et al., 2003, Kashima et al., 2003, Sugiyama and Borgwardt, 2015] and the DBSCAN clustering algorithm [Ester et al., 1997]. In Part 1 you will implement the random walk kernel using python's `networkx` module. In Part 2 you will perform DBSCAN on a toy data set using specified parameters.

## Homework Part 1: Random Walk Graph Kernel

### Introduction

The idea of the random walk kernel is simple: We perform random walks on two given graphs and count how often we performed the same walk. We can use the concept of direct product graphs to walk on two graphs simultaneously.

Let's denote a graph $G = (V, E)$ as a tuple of its vertices $V$ and edges $E$. An edge can be represented as a set of its adjacent vertices $\{u, v\}$, with $u, v \in V$. Furthermore, a vertex $v$ can be attributed with a label $l_v$. A walk $w$ on $G$ consists of $i$ vertices

$w = v_1, v_2, \ldots, v_i$, with $v \in V$ such that each consecutive node can be reached through an edge with its predecessor: $v_i, v_{i+1} \in E$. The direct product graph $G_\times = (V_\times, E_\times)$ of two graphs $G = (V, E)$ and $G' = (V', E')$ is a graph whose vertices are pairs of nodes from $G_1$ and $G_2$

$$V_\times = \{(v_i, v'_{i'}) : v_i \in V, v'_{i'} \in V'\}$$

which are connected if and only if the corresponding nodes in $G$ and $G'$ are neighbors:

$$E_\times = \{((v_i, v'_{i'}), (v_j, v'_{j'})) : (v_i, v_j) \in E \wedge (v'_{i'}, v'_{j'}) \in E'\} \tag{1}$$

Figure 1 depicts how the product graph is constructed.

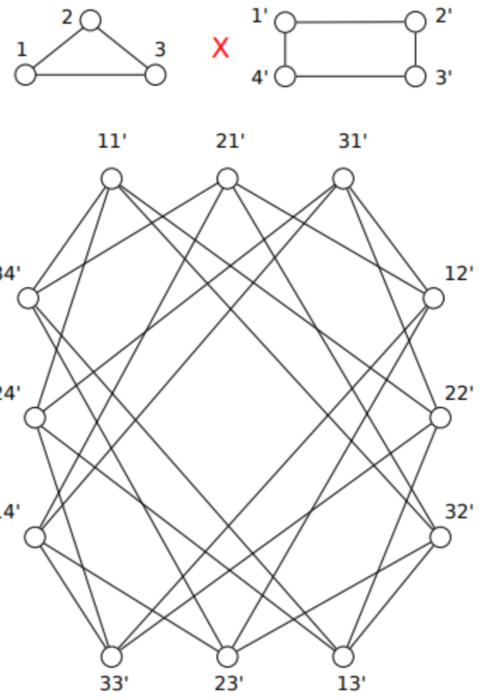

Figure 1: Resulting direct product graph of two labeled graphs. Taken from [Vishwanathan et al., 2009].

From the weight matrix $W_\times$ of the direct product graph, a vector of starting probabilities $q_\times$, stopping probabilities $p_\times$, the identity matrix $\mathbb{1}$, and sufficiently small ([Vishwanathan, 2003]) $\lambda$ we can then define the random walk kernel as follows:

$$k(G, G') = q_\times^T (\mathbb{1} - \lambda W_\times)^{-1} p_\times \tag{2}$$

## Exercise 1

In this exercise you will implement the random walk kernel by utilizing the python module `networkx` (installable via `'pip install networkx'`), a library that allows us to generate and manipulate graphs. The following functions might be helpful to solve the exercise:

```
import networkx as nx
my_graph = nx.Graph() # Generates a new empty graph object
# Adding nodes
```

```
my_graph.add_node("node_1") # Adds node "node_1" to the graph
my_graph.add_nodes_from([1,2,3]) # Adds nodes 1, 2, and 3 to the graph
# Adding edges
my_graph.add_edge((1,2)) # Adds an edge between node 1 and 2
my_graph.add_edges_from([(1,2), (1,3), (2,3)]) # Adds 3 edges
for node in my_graph.nodes(): # Let's you iterate over all nodes
    print(node)
for edge in my_graph.edges(): # Let's you iterate over all edges
    print(edge)
A = nx.to_numpy_matrix(my_graph) # Returns the graph's adjacency matrix
my_graph = nx.from_numpy_matrix(A) # Returns graph from adjacency matrix
```

In this exercise, no arguments are necessary to run your code. All parameters (e.g. $\lambda$) can be 'hard coded' in each script.

**Exercise 1.a Construction of direct product graph** In script `hw_6_ex_1a.py` implement a function called `direct_product_graph(G_1, G_2)` that returns the product graph of two given graphs. Do this in two steps:

1. Create the nodes of the direct product graph. Sticking to the node labels as shown in figure 1 might be helpful to debug your code.
2. Add the edges to the graph that satisfy equation 1

Call this function with the graphs from figure 1 that you have to instantiate in the main function of the script. Do not use any third party library other than `networkx` for this part of the exercise.

**Exercise 1.b Direct product graph via Kronecker product** It can be shown (e.g. [Vishwanathan et al., 2009]) that the weight matrix $W_\times$ of the direct product graph is equal to its normalized adjacency matrix $A_\times$. $A_\times$ can be constructed via the Kronecker product of the normalized adjacency matrices $A_G$ and $A_{G'}$ of the given graphs $G$ and $G'$.
An adjacency matrix $\tilde{A}$ of a Graph $G = (V, E)$ with $i$ nodes is an $i \times i$ binary matrix with

$$\tilde{A}_{ij} = 1 \iff (v_i, v_j) \in E$$
$$\tilde{A}_{ij} = 0 \iff (v_i, v_j) \notin E$$

The normalized form of $\tilde{A}$ can be computed by multiplying it with the reciprocal of its degree matrix $D$ in the following way:

$$A = D^{-1}\tilde{A}$$

$D$ is an $i \times i$ diagonal matrix whose non-zero entries are the numbers of edges attached to the $i$th vertex. We can now calculate $W_\times$ in the following way:

$$W_\times = A_\times = A_G \otimes A_{G'}$$

The Kronecker product is defined for an $m \times n$ matrix $A$ and a $p \times q$ matrix $B$ as follows

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}$$

In script `hw_6_ex_1b.py` implement a function `direct_product_graph_adj_matrix(G_1, G_2)` that normalizes the adjacency matrices of its input and uses their Kronecker product to construct $W_\times$. You can use `numpy` for this part of the exercise.

**Exercise 1.c Random walk kernel** In script `hw_6_ex_1c.py` implement a function `rw_kernel(G_1, G_2, lam)` that

1. Generates the direct product graph and its weight matrix using your function from exercise 1.b
2. Computes the start and stopping probabilities ($q_\times = p_\times$) as the vector of the diagonal elements of $G'_\times s$ degree matrix normalized by the degree sum.
3. Uses 1. and 2. to calculate the random walk kernel as defined in equation 2 and $\lambda = 0.01$

**Exercise 1.d Time complexity** What is the time complexity of the random walk kernel? What makes this naive implementation so "expensive"?

## Homework Part 2: DBSCAN algorithm

### Introduction

Given a distance matrix between all pairs of points in a data set, DBSCAN groups objects as *core points*, *border points* and *noise points* with parameters $\epsilon$ and MinPts. Note that the $\epsilon$-neighborhood of $p$, $N_\epsilon(p)$, is defined as the set of points (including $p$ itself) within a distance of $\epsilon$ of the point $p$. The size of $N_\epsilon(p)$ is denoted by $|N_\epsilon(p)|$.

- Core points: $p$ is a core point iff there are at least MinPts points within its $\epsilon$-neighborhood, i.e.
$$|N_\epsilon(p)| \geq \text{MinPts}$$
- Border points: $p$ is a border point iff it is not a core point and there is at least one core point in its $\epsilon$-neighborhood.
$$|N_\epsilon(p)| < \text{MinPts and } \exists q \in |N_\epsilon(p)| : |N_\epsilon(q)| \geq \text{MinPts}$$
- Noise points: $p$ is a noise point iff it is not a core point or a border point.

Clusters are then formed from the connected core points (there is a path between all the core points in a cluster) and their $\epsilon$-neighborhood (Border points). The clusters are formed one-by-one, thus *the assignment of border points is order-dependent*. Finally, the noise points are reported as outliers.

### Exercise 2

An example data set is shown in Figure 2, where each node represents a point and each edge represents the distance between two points. If there is no direct link between two nodes, their distance is considered as infinite.

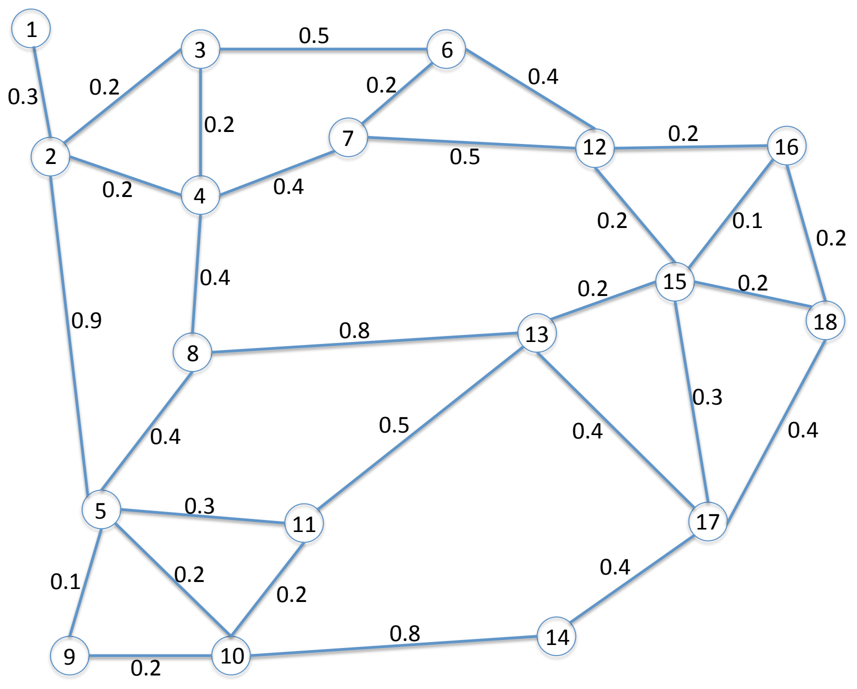

Figure 2: Example data set

**Exercise 2.a** Perform DBSCAN on this data set and indicate the core points, border points, noise points and the clustering assignment (**using the scanning order from node** 1 **to node** 18) for the following parameters:

- $\epsilon = 0.3$ and MinPts $= 4$
- $\epsilon = 0.4$ and MinPts $= 4$
- $\epsilon = 0.4$ and MinPts $= 5$

Compute the results by hand (the file `DBSCAN_graph.pptx` contains Figure 2), the results must be reported in the following format:

```
epsilon=0.3 and MinPts = 4
Core points: 1, 2, 3, 4
Border points: 5, 6, 7, 8
Noise points: 9, 10, 11, 12
Clustering assignment: Cluster1:{1, 2, 3, 4}, ... , ClusterN:{13, 14, 15}
```

**Exercise 2.b** Describe the effect of varying MinPts and $\epsilon$ in Exercise 1.a.

## Grading

This homework is worth a total of 100 points. Table 1 shows the points assigned to each exercise.

Table 1: Grading key for Homework 6

| | | |
|---|---|---|
| **50 pts.** | **Exercise 1** | |
| | 10 pts. | Exercise 1.a |
| | 15 pts. | Exercise 1.b |
| | 15 pts. | Exercise 1.c |
| | 10 pts. | Exercise 1.d |
| **50 pts.** | **Exercise 2** | |
| | 40 pts. | Exercise 2.a |
| | 10 pts. | Exercise 2.b |

## Acknowledgements

This exercise was created by Karsten Borgwardt, Christian Bock, and Xiao He.

## References

M. Ester, H. Kriegel, J. Sander, and X. Xu. Density-connected sets and their application for trend detection in spatial databases. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, California, USA, August 14-17, 1997*, pages 10–15, 1997. URL `http://www.aaai.org/Library/KDD/1997/kdd97-002.php`.

T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In B. Schölkopf and M. K. Warmuth, editors, *Learning Theory and Kernel Machines*, pages 129–143, Heidelberg, Germanz, 2003. Springer.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th International Conference on Machine Learning*, pages 321–328, 2003.

M. Sugiyama and K. Borgwardt. Halting in random walk kernels. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1639–1647. 2015.

S. Vishwanathan. *Kernel Methods Fast Algorithms and real life applications*. PhD thesis, Indian Institute of Science Bangalore, 2003.

S. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 10:1–41, 2009.