Graph Kernels Based on Linear Patterns: Theoretical and Experimental Comparisons

Linlin Jia (linlin.jia@insa-rouen.fr)
Advisors : Benoit Gaüzère & Paul Honeine
LITIS Lab, INSA Rouen & Université de Rouen, Normandy, France

1. Abstract

Graph kernels are powerful tools to bridge the gap between machine learning and data encoded as graphs. Most graph kernels are based on a decomposition of graphs into a set of patterns. The similarity between graphs is then deduced from the similarity of corresponding patterns. Among different possible sets of patterns, linear patterns based kernels often constitute a good trade off between time consumption and accuracy performance.

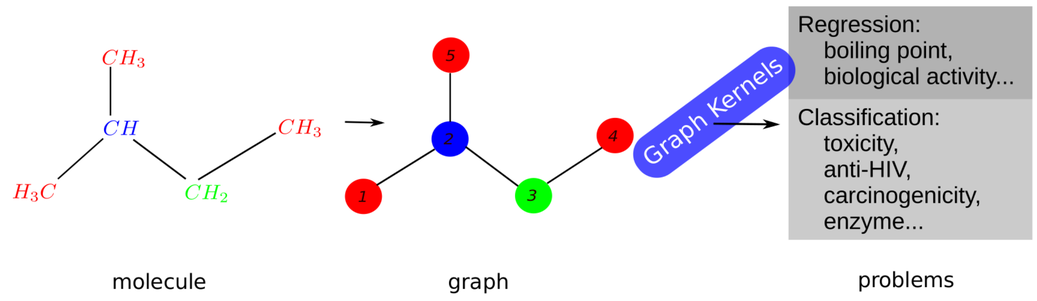

Figure 1: Tasks in chemoinformatics

In this work, we propose a thorough study and comparison of the existing graph kernels based on different linear patterns, namely walks and paths. This work leads to a clear comparison of pros and cons of different proposed kernels.

2. Graph Kernels Based on Linear Patterns

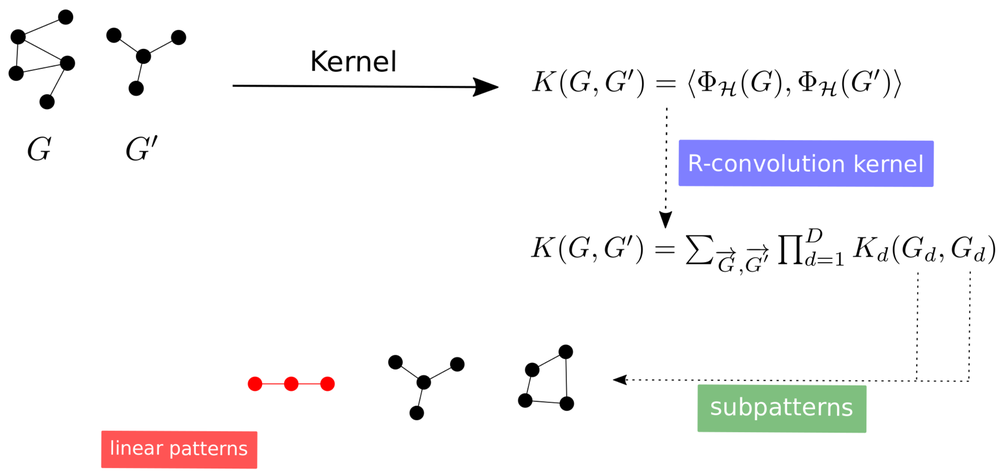

Figure 2: Linear Graph Kernels

- **Common walk kernels**: $K_\times(G_1,G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{h=1}^{\infty} \lambda_h A_\times^h \right]_{ij}$
  - Feature space: all possible walk sequences in graphs
- **Marginalized kernels**: $K(G_1,G_2) = \sum_{w_1 \in W(G_1)} \sum_{w_2 \in W(G_2)} k_W(w_1,w_2) p_{G_1}(w_1) p_{G_2}(w_2)$
  - Feature space: infinite dimensional random walk count vectors
- **Random walk kernels**: $K(G,G') = \sum_{k=0}^{\infty} (k) q_\times^\top W_\times^k p_\times$
  - Feature space: random walks
  - A unified framework of common walk kernels and marginalized kernels
- **Shortest path kernels**: $K_{sp}(S_1,S_2) = \sum_{e_1 \in E_1} \sum_{e_2 \in E_2} k_w(e_1,e_2)$
  - Feature space: all paths of length 1
- **Structural SP kernels**: $K_{ssp}(G_1,G_2) = \frac{1}{n_1}\frac{1}{n_2} \sum_{i:p_i \in P_1} \sum_{j:p_j \in P_2} k_p(p_i,p_j)$
  - Feature space: all shortest paths
- **Path kernels up to length** h: $K_{path}(G_1,G_2) = \sum_{i:p_i \in P_1} \sum_{j:p_j \in P_2} k_p(p_i,p_j)$
  - Feature space: all paths up to length h

Table 1: Comparison of linear graph kernels

Kernels	Substructures			Labeling				Directed	Complexity	Explicit Representation	Weighting
	linear	non-linear	cyclic	symbolic		non-symbolic					
				nodes	edges	nodes	edges				
Common walk	✓	✗	✗	✓	✓	✗	✗	✓	$\mathcal{O}(n^6)$	✗	a priori
Marginalized	✓	✗	✗	✓	✓	✗	✗	✓	polynomial	✗	✗
Random walk	✓	✗	✗	✓	✓	✗	✗	✓	$\mathcal{O}(n^3)$	✗	✗
Shortest path	✓	✗	✗	✓	✗	✓	✗	✓	$\mathcal{O}(n^4)$	✗	✗
Structural shortest path	✓	✗	✗	✓	✓	✓	✓	✓	$\mathcal{O}(\lambda n^4 + nm)$	✗	✗
Path kernel up to length h	✓	✗	✗	✓	✓	✗	✗	✓	$\mathcal{O}(hnm)$	✓	✓

3. Comparison of Linear Graph Kernels

- **On different graph kernels**:
  - Mathematical expressions
  - Time complexity analysis
  - Pros and cons of each kernel
  - Relationships between kernels
  - From walks to paths: tottering and halting problems
- **On different types of graph datasets (accuracy and time complexity)**:
  - Labeled and unlabeled graphs
  - Directed and undirected graphs
  - Graphs with different numbers of nodes
  - Graph with different average degrees
  - Graphs with symbolic and non-symbolic attributes
  - Cyclic and acyclic graphs

Choose graph kernels properly based on properties of datasets

4. Github Library: Implementation

Functions of Graph Kernels

- **Common walk kernels** (commonwalkkernel):
  - Direct product of labeled graphs
  - Two computation methods based on exponential series and geometric series
- **Marginalized kernels** (marginalizedkernel):
  - Remove tottering by graph transformation
- **Random walk kernels** (randomwalkkernel):
  - Five methods to speed up kernel computation: Sylvester equation methods, conjugate gradient methods, fixed-point iterations, spectral decomposition method, nearest Kronecker product approximation.
- **Shortest path kernels** (spkernel):
  - Support non-symbolic node attributes
  - Support missing values of node/edge labels
  - Apply the Fast Computation of Shortest Path Kernel (FCSP) method to speed up computation
- **Path kernels up to length** h (untilhpathkernel):
  - Implement Tanimoto kernel and MinMax kernel
  - Implement suffix tree to speed up (linear time complexity in theory)

Other Functions of Tools

- Methods to retrieve graphs from different dataset structures (loadDataset)
- A method to obtain graph dataset attributes (get_dataset_attributes)
- A model selection function for pre-computed kernel matrices (model_selection_for_precomputed_kernel)

Github Link *https://github.com/jajupmochi/py-graph*

5. Experimental Results

Table 2: Results with minimal test RMSE for each kernel on dataset Acyclics

Kernels	Train Perf	Valid Perf	Test Perf	Parameters	Gram Matrix Time
Common walk	9.38±0.40	14.68±1.15	14.45±4.58	method: 'geo', γ: 0.32, α: 1.00e-08	54.55"±18.30"
Marginalized	12.95±0.37	19.02±1.73	18.24±5.00	p_quit: 0.2, α: 1.00e-04	447.44"±5.32"
Random walk	31.07±0.55	32.09±0.78	32.99±5.18	λ: 0.03, q: 0.9, α: 1.00e-07	7.67"±0.24"
Shortest path	7.43±0.22	10.66±0.54	**10.32±2.38**	α: 0.01	**11.70"**
Structural shortest path	8.71±0.63	19.28±1.75	17.42±6.57	α: 2.82e-02	21.94"
Path kernel up to length h	5.76±0.27	9.89±0.87	**10.21±4.16**	h: 2.0, k_func: 'MinMax', α: 0.1	1.16"±0.75"

- Dataset: A database of acyclic molecules with hetero atoms, consisting of 183 molecules. The task is boiling point prediction.
- Experimental method: A two layer nested cross validation + 90% of the dataset as train set and remaining 10% as test set.