ETH zürich

Module II: Methods to Compare Structured Biomedical Data

Felipe Llinares-López and Damian Roqueiro
Machine Learning & Computational Biology Lab
D-BSSE, ETH Zürich

Tutorial AM2: Machine learning methods in the analysis of genomic and clinical data. July 6, 2018

Structured biomedical data

What makes it "structured"?

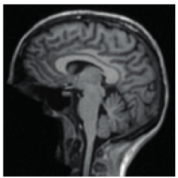
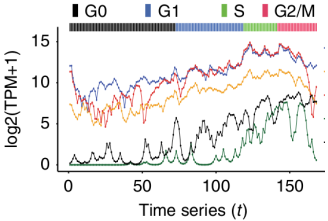


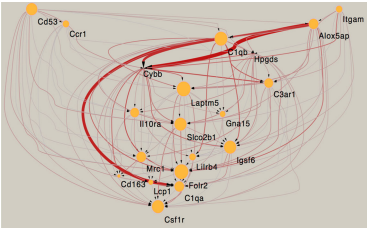
Image: MRI

Zhu et al. [2018]



Single-cell time series

Liu et al. [2017]

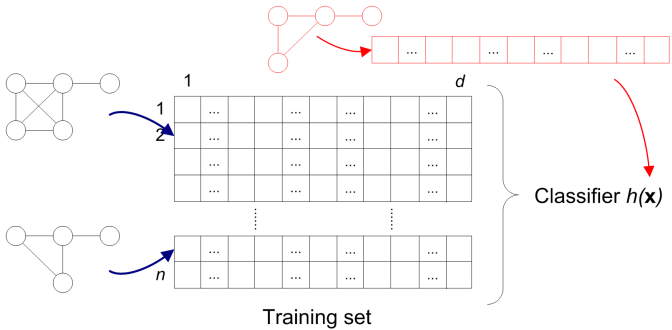


Gene co-expression network

Mueller et al. [2017]

Data transformation

A review



- ML toolboxes expect **data matrix** as *samples* $\times$ *features*
- We must transform the (original) structured data into a vectorial representation

Classification

Root of the problem

Josef Steppan. https://commons.wikimedia.org/w/index.php?curid=64810040

- Idea: to classify we need a **measure of similarity** between objects
- How we measure similarity has direct impact on classification

Classification

Root of the problem

Josef Steppan. https://commons.wikimedia.org/w/index.php?curid=64810040

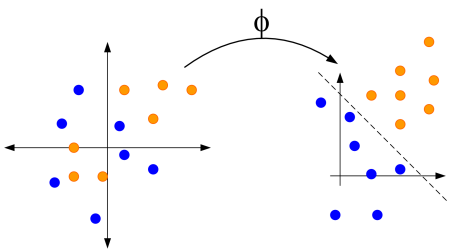
- Idea: to classify we need a **measure of similarity** between objects
- How we measure similarity has direct impact on classification

Part I. Kernels

What is a kernel?
Their properties
Kernelizing an algorithm

What is a kernel?

Intuition



Source: All icons in figures were downloaded & modified from: flaticon.com (designed by Freepik)

- Map two objects $\mathbf{x}$ and $\mathbf{x}'$ onto Hilbert space $\mathcal{H}$ via mapping Φ
- Compute similarity between $\mathbf{x}$ and $\mathbf{x}'$ as inner product $\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathcal{H}}$

- **Kernel trick**: Compute inner product in $\mathcal{H}$ as kernel in input space

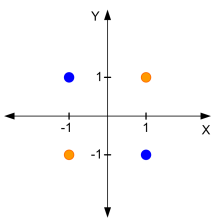
$$k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathcal{H}}$$

What is a kernel?

Example: XOR problem Gärtner [2003]

- Dataset $\mathcal{D}$ with data points in $\mathbb{R}^2$:
  $(+1,+1)$, $(-1,-1)$, $(+1,-1)$, $(-1,+1)$
- Define the mapping
  $\Phi : (x_1, x_2) \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2)$
- Separation is possible with plane orthogonal to $(0,1,0)$
- Explicit mapping

$$\Phi(+1,+1) = (+1,+\sqrt{2},+1)$$
$$\Phi(-1,-1) = (+1,+\sqrt{2},+1)$$
$$\Phi(+1,-1) = (+1,-\sqrt{2},+1)$$
$$\Phi(-1,+1) = (+1,-\sqrt{2},+1)$$

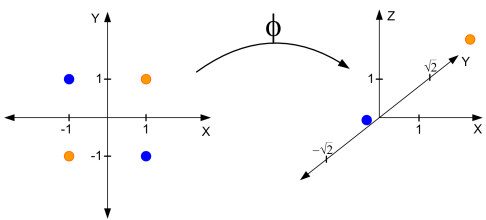


What is a kernel?

Example: XOR problem Gärtner [2003]

- Dataset $\mathcal{D}$ with data points in $\mathbb{R}^2$:
  $(+1,+1)$, $(-1,-1)$, $(+1,-1)$, $(-1,+1)$
- Define the mapping
  $\Phi : (x_1, x_2) \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2)$
- Separation is possible with plane orthogonal to $(0,1,0)$
- Explicit mapping

$$\Phi(+1,+1) = (+1,+\sqrt{2},+1)$$
$$\Phi(-1,-1) = (+1,+\sqrt{2},+1)$$
$$\Phi(+1,-1) = (+1,-\sqrt{2},+1)$$
$$\Phi(-1,+1) = (+1,-\sqrt{2},+1)$$

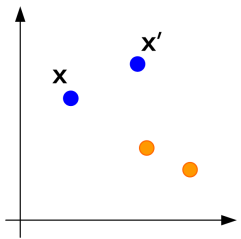


Inner product

Definition

- Let $\mathbf{x}$ and $\mathbf{x}' \in \mathbb{R}^d$
- **Inner product**:

$$\langle \mathbf{x}, \mathbf{x}' \rangle = \mathbf{x}^\top \mathbf{x}' = \sum_{i=1}^{d} x_i x_i'$$



- Geometric interpretation:

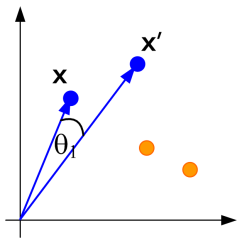
$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle = \|\mathbf{x}\| \|\mathbf{x}'\| \cos\theta_1$$
$$k(\mathbf{x}, \mathbf{x}'') = \langle \mathbf{x}, \mathbf{x}'' \rangle = \|\mathbf{x}\| \|\mathbf{x}''\| \cos\theta_2$$

Inner product

Definition

- Let $\mathbf{x}$ and $\mathbf{x}' \in \mathbb{R}^d$
- **Inner product**:

$$\langle \mathbf{x}, \mathbf{x}' \rangle = \mathbf{x}^\top \mathbf{x}' = \sum_{i=1}^{d} x_i x_i'$$



- Geometric interpretation:

$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle = \|\mathbf{x}\| \|\mathbf{x}'\| \cos\theta_1$$

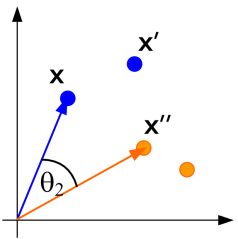
$$k(\mathbf{x}, \mathbf{x}'') = \langle \mathbf{x}, \mathbf{x}'' \rangle = \|\mathbf{x}\| \|\mathbf{x}''\| \cos\theta_2$$

Inner product

Definition

- Let $\mathbf{x}$ and $\mathbf{x}' \in \mathbb{R}^d$
- **Inner product**:

$$\langle \mathbf{x}, \mathbf{x}' \rangle = \mathbf{x}^\top \mathbf{x}' = \sum_{i=1}^{d} x_i x_i'$$



- Geometric interpretation:

$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle = \|\mathbf{x}\| \|\mathbf{x}'\| \cos\theta_1$$
$$k(\mathbf{x}, \mathbf{x}'') = \langle \mathbf{x}, \mathbf{x}'' \rangle = \|\mathbf{x}\| \|\mathbf{x}''\| \cos\theta_2$$

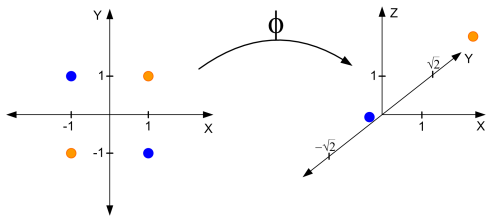
Kernel

Revisiting the XOR problem Gärtner et al. [2003]

- Recall the mapping
  $\Phi : (x_1, x_2) \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

- Implicit transformation with kernel:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= \langle \mathbf{x}, \mathbf{x}' \rangle^2 \\
&= \langle (x_1, x_2), (x_1', x_2') \rangle^2 = (x_1x_1' + x_2x_2')^2 \\
&= (x_1x_1')^2 + 2x_1x_2x_1'x_2' + (x_2x_2')^2 \\
&= \langle (x_1^2, \sqrt{2}x_1x_2, x_2^2), (x_1'^2, \sqrt{2}x_1'x_2', x_2'^2) \rangle \\
&= \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle
\end{aligned}
$$



Kernels

- Linear kernel

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} x_i x_i' = \mathbf{x}^\top \mathbf{x}'$$

- Polynomial kernel (of degree p)

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + c)^p \qquad \text{with } c \in \mathbb{R}, p \in \mathbb{N}^+$$

- Gaussian RBF kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\Big(-\frac{1}{2\sigma^2} \|\mathbf{x} - \mathbf{x}'\|^2 \Big) \qquad \text{with } \sigma \in \mathbb{R}$$

Implicit vector embedding

Kernel trick

$$k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathcal{H}}$$

- For all data points $\mathbf{x}$ and $\mathbf{x}'$
- Take an algorithm $\mathcal{A}$ defined in terms of inner products between $\mathbf{x}$ and $\mathbf{x}'$
- Substitute the inner products with a kernel $k(\mathbf{x}, \mathbf{x}')$
- Obtain kernelized version of $\mathcal{A}$

k-Nearest neighbor

Algorithm

procedure k-NN($\mathcal{X}_{train}$, $\mathcal{Y}_{train}$, k, $\mathbf{x}_{test}$)
▷ $\mathcal{X}_{train}, \mathcal{Y}_{train}$: Data points, labels in training set
▷ k: Number of neighbors
▷ $\mathbf{x}_{test}$: Data point to predict label

▷ Neighborhood
  $\mathcal{N} \leftarrow \emptyset$
  for each $\mathbf{x} \in \mathcal{X}_{train}$ **do**
    $d \leftarrow$ `get_distance`$(\mathbf{x}, \mathbf{x}_{test})$
    $\mathcal{N} \leftarrow$ `update_neighborhood`$(\mathcal{N}, k, (\mathbf{x}, d))$
▷ Predict the label based on $\mathcal{N}$
  $\hat{y} \leftarrow$ `majority_vote`$(\mathcal{N}, \mathcal{Y}_{train})$

function `get_distance`$(\mathbf{x}, \mathbf{z})$
▷ Compute the Euclidean distance
  $dist \leftarrow 0$
  for $i \leftarrow 1, d$ **do**
    $dist \leftarrow dist + (x_i - z_i)^2$
  return $\sqrt{dist}$

Euclidean distance

As inner product

$$\begin{aligned}\|\mathbf{x}-\mathbf{z}\|^2 &= \sum_{i=1}^{d}(x_i - z_i)^2 = \sum_{i=1}^{d}(x_i^2 + z_i^2 - 2x_i z_i) \\ &= \sum_{i=1}^{d} x_i^2 + \sum_{i=1}^{d} z_i^2 - \sum_{i=1}^{d} 2x_i z_i \\ &= \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{z}, \mathbf{z} \rangle - 2\langle \mathbf{x}, \mathbf{z} \rangle\end{aligned}$$

Implicit transformation with **kernel**

$$\begin{aligned}\|\mathbf{x}-\mathbf{z}\|^2 = \|\Phi(\mathbf{x}) - \Phi(\mathbf{z})\|^2 &= \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}) \rangle + \langle \Phi(\mathbf{z}), \Phi(\mathbf{z}) \rangle - 2\langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle \\ &= k(\mathbf{x}, \mathbf{x}) + k(\mathbf{z}, \mathbf{z}) - 2k(\mathbf{x}, \mathbf{z})\end{aligned}$$

Kernelized k-Nearest neighbor

Possible implementation

procedure k-NN($\mathcal{X}_{train}$, $\mathcal{Y}_{train}$, k, $\mathbf{x}_{test}$)
▷ $\mathcal{X}_{train}, \mathcal{Y}_{train}$: Data points, labels in training set
▷ k: Number of neighbors
▷ $\mathbf{x}_{test}$: Data point to predict label

▷ Neighborhood
  $\mathcal{N} \leftarrow \emptyset$
  for each $\mathbf{x} \in \mathcal{X}_{train}$ **do**
    $s \leftarrow$ `get_similarity`$(\mathbf{x}, \mathbf{x}_{test})$
    $\mathcal{N} \leftarrow$ `update_neighborhood`$(\mathcal{N}, k, (\mathbf{x}, s))$
▷ Predict the label based on $\mathcal{N}$
  $\hat{y} \leftarrow$ `majority_vote`$(\mathcal{N}, \mathcal{Y}_{train})$

function `get_similarity`$(\mathbf{x}, \mathbf{z})$
▷ Use a kernel
  return $k(\mathbf{x}, \mathbf{x}) + k(\mathbf{z}, \mathbf{z}) - 2k(\mathbf{x}, \mathbf{z})$

Kernelized k-Nearest neighbor

Possible kernels

- **Linear**: $k(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle$

  Defaults to standard (squared) Euclidean distance

- **Polynomial**: $k(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + 1)^3$

  Shown to perform better than original k-NN in liver disorder database Yu et al. [2002]

function `get_similarity`$(\mathbf{x}, \mathbf{z})$
▷ Use a kernel
  return $k(\mathbf{x}, \mathbf{x}) + k(\mathbf{z}, \mathbf{z}) - 2k(\mathbf{x}, \mathbf{z})$

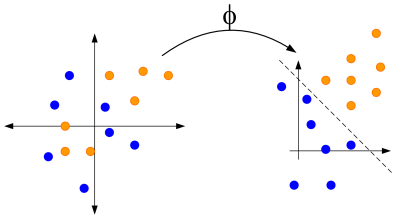
Kernel

Definition Hofmann et al. [2008]

- For $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ define $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ as

$$k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$$

where Φ maps into an inner product space $\mathcal{H}$



- Given a dataset $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$ the **Gram matrix** $\mathbf{K}$ is defined as

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \ldots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & \ldots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \ldots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

Properties of kernels

Closure Shawe-Taylor and Cristianini [2004]

Assume k_1 and k_2 are kernels over $\mathcal{X} \times \mathcal{X}$

$\mathcal{X} \in \mathbb{R}^d$

$\Phi : \mathcal{X} \mapsto \mathbb{R}^m$

- $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$
- $k(\mathbf{x}, \mathbf{x}') = a\, k_1(\mathbf{x}, \mathbf{x}')$, with $a \in \mathbb{R}^+$
- $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}')$
- $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{B} \mathbf{x}'$, with $\mathbf{B} \in \mathbb{R}^{d \times d}$ symmetric and positive semi-definite

Part II. Kernels on structured data

String kernels
Graph kernels

Kernels on structured data

R-convolution kernels Haussler [1999]

- **Intuition**: Decompose two objects X and X' into sets of substructures S and S'.
- The idea is to compare all pairs of the substructures of X and X':

$$k_R(X, X') = \sum_{s \in S, s' \in S'} k_{base}(s, s')$$

- For example, a substructure could be the elements of a set, the nodes of a graph or the substrings of a string.
- k_{base} is an arbitrary vectorial kernel, very often even the Dirac delta kernel.

Motivation

Comparison of strings

Protein binds	No binding
ACTGGCA	GCATTGCTG
TTTCGAA	AGTGATC
GTAGGAA	CGCATT
CCTGGTACA	CCGGTAC

- Assume we have training data with two sets of labeled DNA short sequences
- **Question**: To which sequence(s) is a new one most similar?

Motivation

Comparison of strings

Protein binds	No binding
ACTGGCA	GCATTGCTG
TTTCGAA	AGTGATC
GTAGGAA	CGCATT
CCTGGTACA	CCGGTAC

? TCGGCATT

- Assume we have training data with two sets of labeled DNA short sequences
- **Question**: To which sequence(s) is a new one most similar?

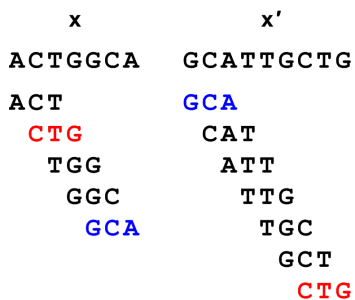
Spectrum kernel (k-mers)

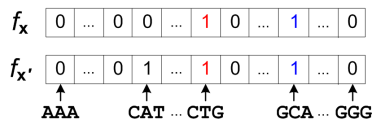
Approach Leslie et al. [2002]

- Let $\mathcal{A}$ be the underlying alphabet $\rightarrow \mathcal{A}^k$ all strings of length k
- Index the feature space by k-length strings in $\mathcal{A}^k$
- For each $s \in \mathcal{A}^k$, count separately the occurrences in $\mathbf{x}$ and $\mathbf{x}'$ in $f_{\mathbf{x}}$ and $f_{\mathbf{x}'}$, respectively

- The kernel between two strings $\mathbf{x}$ and $\mathbf{x}'$ is defined as:

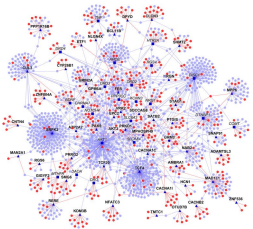
$$k(\mathbf{x}, \mathbf{x}') = \langle f_{\mathbf{x}}, f_{\mathbf{x}'} \rangle$$



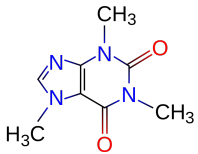


Networks

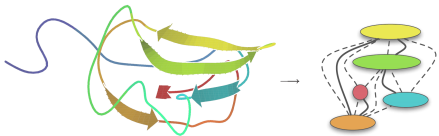
In all walks of life



PPI network
https://commons.wikimedia.org



Chemical compound
https://commons.wikimedia.org

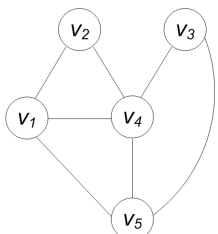


Protein structure
Vishwanathan et al. [2010]

Graphs

Some terminology

- A graph $G = (V, E)$ is a set V of vertices and a set E of edges
- Our focus will be on **undirected** graphs, i.e. edge $e_{ij} = (v_i, v_j)$ is unordered
- A graph is **labeled** if $v_i \in V$ has labels in $\mathcal{L}_V$ and/or $e_{ij} \in E$ has labels in $\mathcal{L}_E$
- Two vertices v_i and v_j are adjacent if there is an edge (v_i, v_j) connecting them

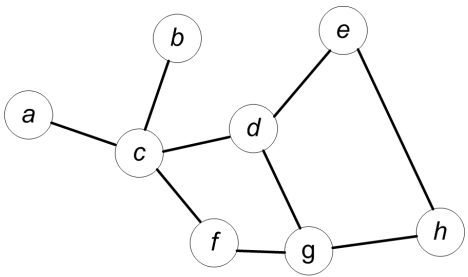


G

- A **walk** $\mathcal{W}$ is a sequence of vertices $\mathcal{W} = v_1, v_2, \ldots, v_k$, where v_i is adjacent to v_{i+1}
- A **path** $\mathcal{P}$ is a walk where $v_i \neq v_j, \forall i \neq j$

Graph representation

Adjacency matrix

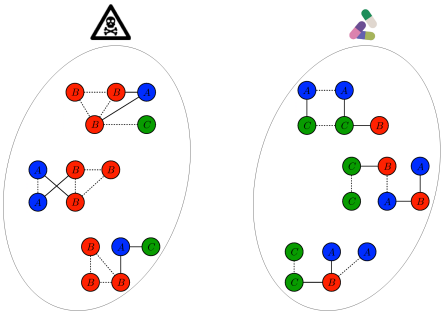


G

$$
\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array}
\begin{array}{c}
\begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\
\left(\begin{array}{cccccccc}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{array}\right)
\end{array}
$$

Motivation

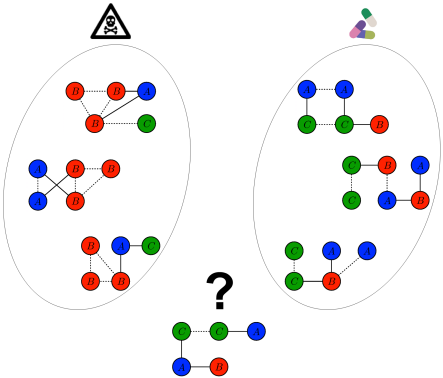
Comparison of graphs



- Assume we have training data with two sets of labeled molecules
- **Key question**: How to measure similarity between graphs?

Motivation

Comparison of graphs

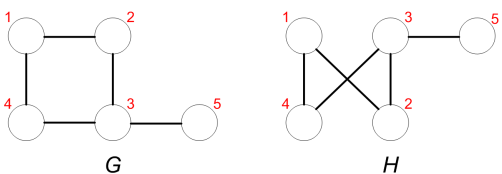


- Assume we have training data with two sets of labeled molecules
- **Key question**: How to measure similarity between graphs?

Why is it difficult to compare graphs?

Graph isomorphism

- Two graphs G and H are **isomorphic**,
  - if there exists a bijection $f : V(G) \mapsto V(H)$
  - such that, for every $(u, v) \in E(G) \Leftrightarrow (f(u), f(v)) \in E(H)$
- Complexity: Claimed to be solvable in quasi-polynomial time Babai [2015]



Subgraph isomorphism

Subgraph isomorphism

- Given two graphs G and H
  - determine whether a subgraph in G is isomorphic to H
  - determine the size of the largest common subgraph in G and H
- Shown to be NP-complete Garey and Johnson [1990]

Implications of NP-completeness

- Runtime may grow exponentially with the number of nodes
- For large graphs (many nodes) and for large datasets of graphs this can be a serious problem

Subgraph isomorphism

Subgraph isomorphism

- Given two graphs G and H
  - determine whether a subgraph in G is isomorphic to H
  - determine the size of the largest common subgraph in G and H
- Shown to be NP-complete Garey and Johnson [1990]

Implications of NP-completeness

- Runtime may grow exponentially with the number of nodes
- For large graphs (many nodes) and for large datasets of graphs this can be a serious problem

Graph/subgraph isomorphism

Special cases

- Graph and subgraph isomorphism are proved to be solvable in linear time for:
  - Trees
  - Planar graphs
  - Interval graphs
  - and others

- Yet, we will focus on the comparison of graphs in general

Kernels on graphs

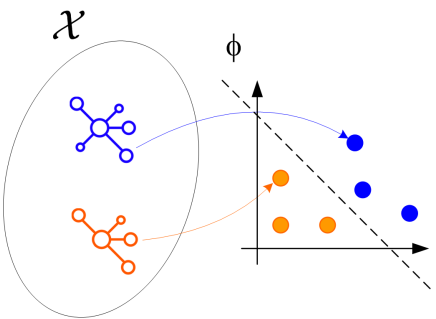
Steps

- Map objects from $\mathcal{X}$ to Hilbert space $\mathcal{H}$

$$\Phi : \mathcal{X} \mapsto \mathcal{H}$$

- Compute similarity between G and G' in $\mathcal{H}$

$$\begin{aligned} k(G, G') &= \langle \Phi(G), \Phi(G') \rangle_{\mathcal{H}} \\ &= \Phi(G)^\top \Phi(G') \end{aligned}$$

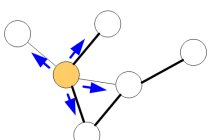


Kernels on graphs

Categories of graph kernels we will discuss today

- Random walks
- Paths
- Limited size sub-graphs
- Sub-tree patterns

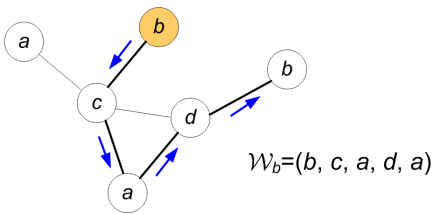
Random walk kernels



Random walk kernel

Characteristics

- $p_s(v_1)$ to select the starting vertex
- In ith. step
  - $p_t(v_i|v_{i-1})$ to decide next vertex
  - $p_e(v_{i-1})$ to halt
- If no prior knowledge $\rightarrow$ uniform probability



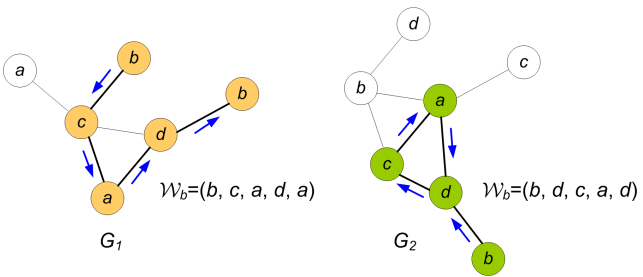
- In general, a walk $\mathcal{W}$ rooted at v_1 is defined as

$$\mathcal{W}_{v_1} = (v_1, e_{v_1,v_2}, v_2, e_{v_2,v_3}, \ldots, v_k)$$

Random walk kernel

Main idea

- Given a pair of graphs, perform random walks on both
- Count the number of "matching" walks



Random walk kernel

Measuring similarity of walks

- Define the similarity between the graphs as similarity between walks

$$k(G, G') = \sum_{\mathcal{W} \in G} \sum_{\mathcal{W}' \in G'} k_{\mathcal{W}}(\mathcal{W}, \mathcal{W}')$$

- Dirac delta $k_{\mathcal{W}}(\mathcal{W}, \mathcal{W}') = \begin{cases} 1 \text{ if } \mathcal{W} = \mathcal{W}' \\ 0 \text{ otherwise} \end{cases}$ Gärtner et al. [2003]

Alternative walk kernels Borgwardt et al. [2005]

- Define $k_{\mathcal{W}}(\mathcal{W}, \mathcal{W}')$ in terms of three kernels
  - i type kernel
  - ii length kernel
  - iii node label kernel

Random walk kernel

Challenges

- Computationally demanding $O(n^6)$
- Tottering: revisiting a vertex immediately after leaving it
- Halting: downweight of longer walks such that the similarity score is dominated by walks of length 1 Sugiyama and Borgwardt [2015]

Paliative measures

- Fast computation of random walk kernels Vishwanathan et al. [2006]
- Adjustment of transition probabilities to prevent tottering

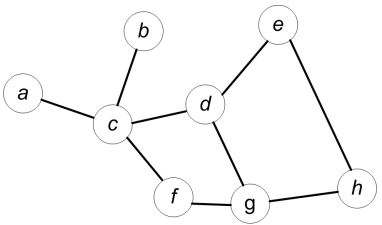
Random walk kernel

Key insigths

- **1.** Walk of length $k \rightarrow$ look at kth. power of adjacency matrix

  Original adjacency matrix $\mathbf{A} = \mathbf{A}^0$

$$\mathbf{A} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



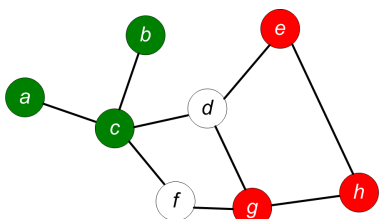
G

Random walk kernel

Key insights

- **1.** Walk of length $k \rightarrow$ look at kth. power of adjacency matrix

  Walks of length $k = 2$ in $\mathbf{A}^2$



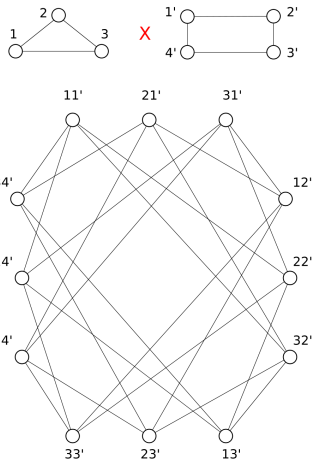
G

$$\mathbf{A}^2 = \begin{array}{c} \\ \mathbf{a} \\ b \\ c \\ d \\ \mathbf{e} \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{cccccccc} a & \mathbf{b} & c & d & e & f & \mathbf{g} & h \end{array} \\ \begin{pmatrix} 1 & \mathbf{1} & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 & 1 & 0 & 2 & 0 \\ 1 & 1 & 0 & 3 & 0 & 2 & 0 & 2 \\ 0 & 0 & 1 & 0 & 2 & 0 & \mathbf{2} & 0 \\ 1 & 1 & 0 & 2 & 0 & 2 & 0 & 1 \\ 0 & 0 & 2 & 0 & 2 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 & 0 & 1 & 0 & 2 \end{pmatrix} \end{array}$$

Random walk kernel

Key insights Vishwanathan et al. [2010]

- **2.** Use direct product graph $G_\times$



Given two graphs $G = (V, E)$ and $G' = (V', E')$

$G_\times$ contains vertex set $V_\times$ and edge set $E_\times$

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

$$E_\times = \{((v_i, v'_r), (v_j, v'_s)) : (v_i, v_j) \in E, (v'_r, v'_s) \in E'\}$$

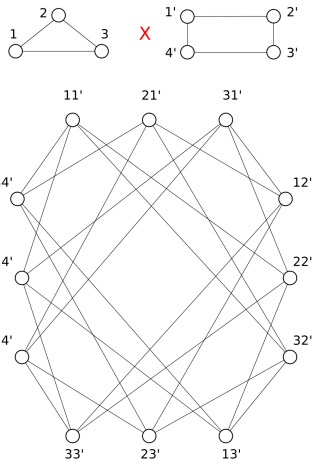
Intuition Random walk on $G_\times$ is simultaneous random walk on G and G'

$$k(G, G') = \sum_{i=1}^{|V|} \sum_{j=1}^{|V'|} \sum_{k=0}^{\infty} \lambda_k [\mathbf{A}_\times^k]_{ij}$$

Random walk kernel

Key insights Vishwanathan et al. [2010]

- **2.** Use direct product graph $G_\times$



Given two graphs $G = (V, E)$ and $G' = (V', E')$

$G_\times$ contains vertex set $V_\times$ and edge set $E_\times$

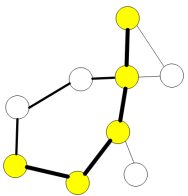
$$V_\times = \{(v_i, v'_r) : v_i \in V, v'_r \in V'\}$$

$$E_\times = \{((v_i, v'_r), (v_j, v'_s)) : (v_i, v_j) \in E, (v'_r, v'_s) \in E'\}$$

Intuition Random walk on $G_\times$ is simultaneous random walk on G and G'

$$k(G, G') = \sum_{i=1}^{|V|} \sum_{j=1}^{|V'|} \sum_{k=0}^{\infty} \lambda_k [\mathbf{A}_\times^k]_{ij}$$

Path-based kernels



Shortest path kernel

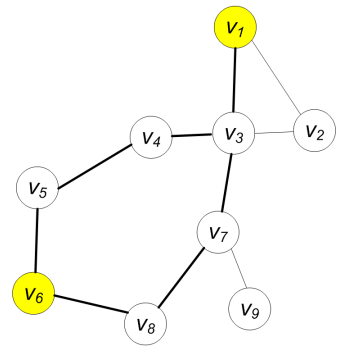
Intuition

- Define a graph kernel based on paths
- Avoid drawbacks from random walk kernels: tottering and halting

Considerations

- Finding *all paths* in a graph is NP-hard
- Finding the *longest paths* is NP-hard
- Finding the **shortest paths** can be done in $O(n^3)$

- Shortest paths may not be unique
- Define kernel based on shortest path distances



Shortest path kernel

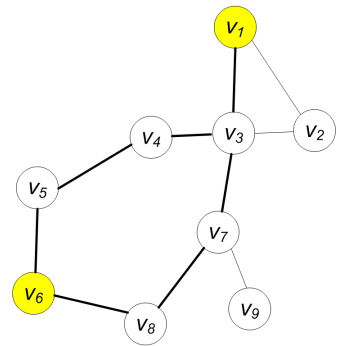
Intuition

- Define a graph kernel based on paths
- Avoid drawbacks from random walk kernels: tottering and halting

Considerations

- Finding *all paths* in a graph is NP-hard
- Finding the *longest paths* is NP-hard
- Finding the **shortest paths** can be done in $O(n^3)$

- Shortest paths may not be unique
- Define kernel based on shortest path distances



Shortest path kernel

Steps Borgwardt and Kriegel [2005]

- Run Floyd-Warshall algorithm to compute all-pairs shortest paths for G and G'

  For directed graphs $\rightarrow$ edge weights $\in \mathbb{R}$ and no negative cycles

  For undirected graphs $\rightarrow$ edge weights $\in \mathbb{R}^+$

- Define a kernel to compare all pairs of shortest path lengths from G and G'

$$k(G, G') = \sum_{v_i, v_j \in V} \sum_{v_r, v_s \in V'} k_{\text{length}}(d(v_i, v_j), d(v_r, v_s))$$

with $d(v_i, v_j)$ the length of the shortest path between v_i and v_j

Shortest path kernel

Kernel for lengths Borgwardt and Kriegel [2005]

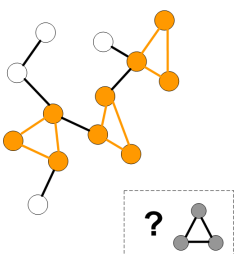
$$k(G, G') = \sum_{v_i, v_j \in V} \sum_{v_r, v_s \in V'} k_{\text{length}}(d(v_i, v_j), d(v_r, v_s))$$

- $k_{\text{length}}(\cdot, \cdot)$ is kernel to compare the lengths of two shortest paths. Possible implementations:

  A linear kernel $k_{\text{length}}(d(v_i, v_j), d(v_r, v_s)) = d(v_i, v_j) d(v_r, v_s)$

  A Dirac delta kernel $k_{\text{length}}(d(v_i, v_j), d(v_r, v_s)) = \begin{cases} 1 \text{ if } d(v_i, v_j) = d(v_r, v_s) \\ 0 \text{ otherwise} \end{cases}$

Kernels based on limited-size subgraphs



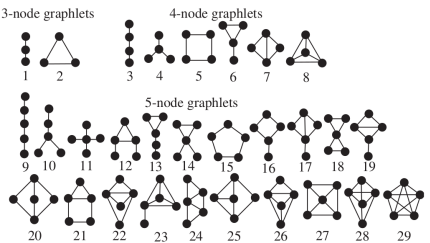
Graphlet kernels

Intuition

- **Graphlet**: *small connected non-isomorphic induced subgraphs* Pržulj [2007]
- Similarity of graphlet distributions ⇒ similarity between corresponding graphs

Challenge

- Counting graphlets of size k takes $O(n^k)$
- Very computationally demanding



Pržulj [2007]

Graphlet kernels

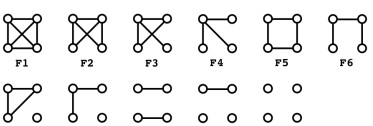
Approach Shervashidze et al. [2009]

- Define g_G^k = count of graphlets of size k in G
- Normalize the counts $n_G^k = \frac{1}{\#\text{ all graphlets in G}}\, g_G^k$
- Define $\mathbf{f}_G \in \mathbb{R}^K$ containing the normalized frequencies
  $\rightarrow$ the ith. entry in $\mathbf{f}_G = n_G^i$

$$k(G, G') = \langle \mathbf{f}_G, \mathbf{f}_{G'} \rangle$$

Considerations

- Perform random sampling
- Consider graphlets of size $k \in \{3, 4, 5\}$



Shervashidze et al. [2009]

Subtree-based kernels

Weisfeiler-Lehman kernel

Algorithm Shervashidze et al. [2011]

function WL_iteration($i, G, G', \mathcal{H}(\cdot)$)

▷ Step 1. Represent each node as sorted list of neighbors

  for each $v \in G$ and G' **do**

    $\mathcal{L}_v \leftarrow \texttt{sort}(\mathcal{N}(v))$

    $s_v \leftarrow \texttt{to\_string}(\mathcal{L}_v)$

▷ Step 2. Compress list of neighbors into hash value

  for each s_v **do**

    $h_v \leftarrow \mathcal{H}(s_v)$

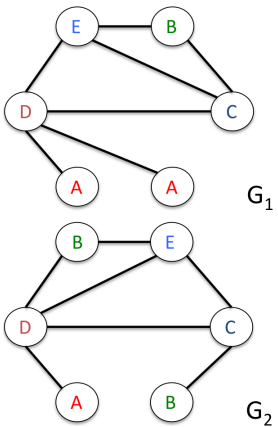
▷ Step 3. Relabel nodes

  for each $v \in G$ and G' **do**

    $label(v) \leftarrow h(v)$

▷ Step 4. Compute the kernel

  return $k(G, G')$



Weisfeiler-Lehman kernel

Algorithm Shervashidze et al. [2011]

function `WL_iteration`$(i, G, G', \mathcal{H}(\cdot))$

▷ Step 1. Represent each node as sorted list of neighbors

  for each $v \in G$ and G' **do**

    $\mathcal{L}_v \leftarrow \texttt{sort}(\mathcal{N}(v))$

    $s_v \leftarrow \texttt{to\_string}(\mathcal{L}_v)$

▷ Step 2. Compress list of neighbors into hash value

  for each s_v **do**

    $h_v \leftarrow \mathcal{H}(s_v)$

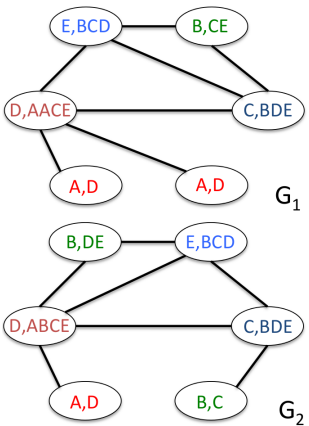
▷ Step 3. Relabel nodes

  for each $v \in G$ and G' **do**

    $label(v) \leftarrow h(v)$

▷ Step 4. Compute the kernel

  return $k(G, G')$



Weisfeiler-Lehman kernel

Algorithm Shervashidze et al. [2011]

function `WL_iteration`$(i, G, G', \mathcal{H}(\cdot))$

▷ Step 1. Represent each node as sorted list of neighbors

  for each $v \in G$ and G' **do**

    $\mathcal{L}_v \leftarrow \texttt{sort}(\mathcal{N}(v))$

    $s_v \leftarrow \texttt{to\_string}(\mathcal{L}_v)$

▷ Step 2. Compress list of neighbors into hash value

  for each s_v **do**

    $h_v \leftarrow \mathcal{H}(s_v)$

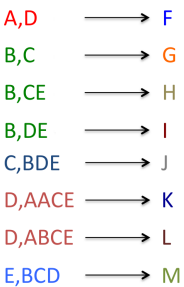
▷ Step 3. Relabel nodes

  for each $v \in G$ and G' **do**

    $label(v) \leftarrow h(v)$

▷ Step 4. Compute the kernel

  return $k(G, G')$



Weisfeiler-Lehman kernel

Algorithm Shervashidze et al. [2011]

function `WL_iteration`$(i, G, G', \mathcal{H}(\cdot))$

▷ Step 1. Represent each node as sorted list of neighbors

  for each $v \in G$ and G' **do**

    $\mathcal{L}_v \leftarrow \texttt{sort}(\mathcal{N}(v))$

    $s_v \leftarrow \texttt{to\_string}(\mathcal{L}_v)$

▷ Step 2. Compress list of neighbors into hash value

  for each s_v **do**

    $h_v \leftarrow \mathcal{H}(s_v)$

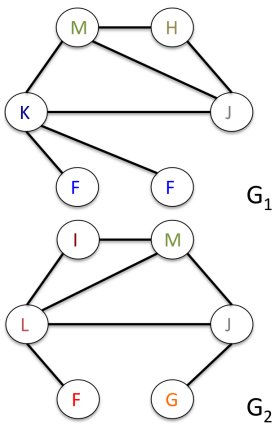
▷ Step 3. Relabel nodes

  for each $v \in G$ and G' **do**

    $label(v) \leftarrow h(v)$

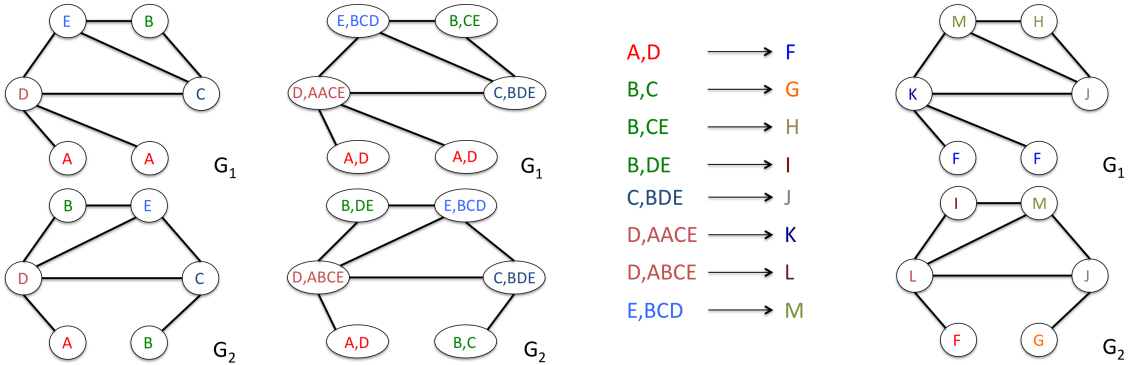
▷ Step 4. Compute the kernel

  return $k(G, G')$



Weisfeiler-Lehman kernel

Algorithm: Iteration 1, steps 1–3 Shervashidze et al. [2011]



Iteration 0 | Step 1 (neighbors) | Step 2 (compression) | Step 3 (relabeling)

Weisfeiler-Lehman kernel

Computing the kernel for m iterations

- For any basic kernel, we have

$$k_{WL}^{m}(G, G') = k(G_0, G_0') + k(G_1, G_1') + \ldots + k(G_m, G_m')$$

- Assume $k(\cdot, \cdot)$ counts pairs of nodes with matching labels
  $k(G, G') = \langle \Phi(G), \Phi(G') \rangle = \sum_{v \in V} \sum_{v' \in V'} \delta(label(v), label(v'))$

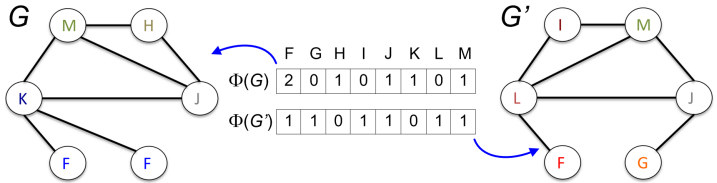
Weisfeiler-Lehman kernel

Computing the kernel for m iterations

- For any basic kernel, we have

$$k_{WL}^{m}(G, G') = k(G_0, G_0') + k(G_1, G_1') + \ldots + k(G_m, G_m')$$

- Assume $k(\cdot, \cdot)$ counts pairs of nodes with matching labels
  $k(G, G') = \langle \Phi(G), \Phi(G') \rangle = \sum_{v \in V} \sum_{v' \in V'} \delta(label(v), label(v'))$



Conclusions contd.

Kernels

- Kernel methods are an efficient tool to compare structured data
- Gram matrices are easily plugged in into ML toolboxes, e.g. SciKit learn
- Graph kernels focus on **how** to compute and compare graph features efficiently

Acknowledgements

Machine Learning and Computational Biology Lab

References I

L. Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015. URL http://arxiv.org/abs/1512.03547.

K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 74–81, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2278-5. doi: 10.1109/ICDM.2005.132. URL http://dx.doi.org/10.1109/ICDM.2005.132.

K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. Vishwanathan, A. J. Smola, and H. P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21 Suppl 1:47–56, Jun 2005.

M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0716710455.

T. Gärtner. A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, 5(1):49–58, July 2003. ISSN 1931-0145. doi: 10.1145/959242.959248. URL http://doi.acm.org/10.1145/959242.959248.

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, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

M. Ghandi, D. Lee, M. Mohammad-Noori, and M. A. Beer. Enhanced regulatory sequence prediction using gapped k-mer features. *PLOS Computational Biology*, 10(7):1–15, 07 2014. doi: 10.1371/journal.pcbi.1003711. URL https://doi.org/10.1371/journal.pcbi.1003711.

References II

D. Haussler. Convolution kernels on discrete structures. In *UCSC-CRL-99-10*, 1999.

T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3): 1171–1220, 2008. ISSN 00905364.

C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: a string kernel for SVM protein classification. *Pac Symp Biocomput*, pages 564–575, 2002.

C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, Mar 2004.

Z. Liu, H. Lou, K. Xie, H. Wang, N. Chen, O. M. Aparicio, M. Q. Zhang, R. Jiang, and T. Chen. Reconstructing cell cycle pseudo time-series via single-cell transcriptome data. *Nat Commun*, 8(1):22, 06 2017.

A. Morrow, V. Shankar, D. Petersohn, A. Joseph, B. Recht, and N. Yosef. Convolutional kitchen sinks for transcription factor binding site prediction, 2017.

A. J. Mueller, E. G. Canty-Laird, P. D. Clegg, and S. R. Tew. Cross-species gene modules emerge from a systems biology approach to osteoarthritis. *npj Systems Biology and Applications*, 3(1):13, 2017. ISSN 2056-7189. doi: 10.1038/s41540-017-0014-3. URL `https://doi.org/10.1038/s41540-017-0014-3`.

N. Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–183, Jan 2007.

References III

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521813972.

N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In D. van Dyk and M. Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL `http://proceedings.mlr.press/v5/shervashidze09a.html`.

N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, Nov. 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2078187`.

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. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5688-halting-in-random-walk-kernels.pdf`.

S. V. N. Vishwanathan, K. M. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 1449–1456, Cambridge, MA, USA, 2006. MIT Press. URL `http://dl.acm.org/citation.cfm?id=2976456.2976638`.

References IV

S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242, Aug. 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1859891.

K. Yu, L. Ji, and X. Zhang. Kernel nearest-neighbor algorithm. *Neural Processing Letters*, 15(2):147–156, Apr 2002. ISSN 1573-773X. doi: 10.1023/A:1015244902967. URL https://doi.org/10.1023/A:1015244902967.

B. Zhu, J. Z. Liu, S. F. Cauley, B. R. Rosen, and M. S. Rosen. Image reconstruction by domain-transform manifold learning. *Nature*, 555:487 EP –, Mar 2018. URL http://dx.doi.org/10.1038/nature25988.