Kernels

Li, Jia

DSAA 5002

The Hong Kong of Science and Technology (Guangzhou)

2025 Fall

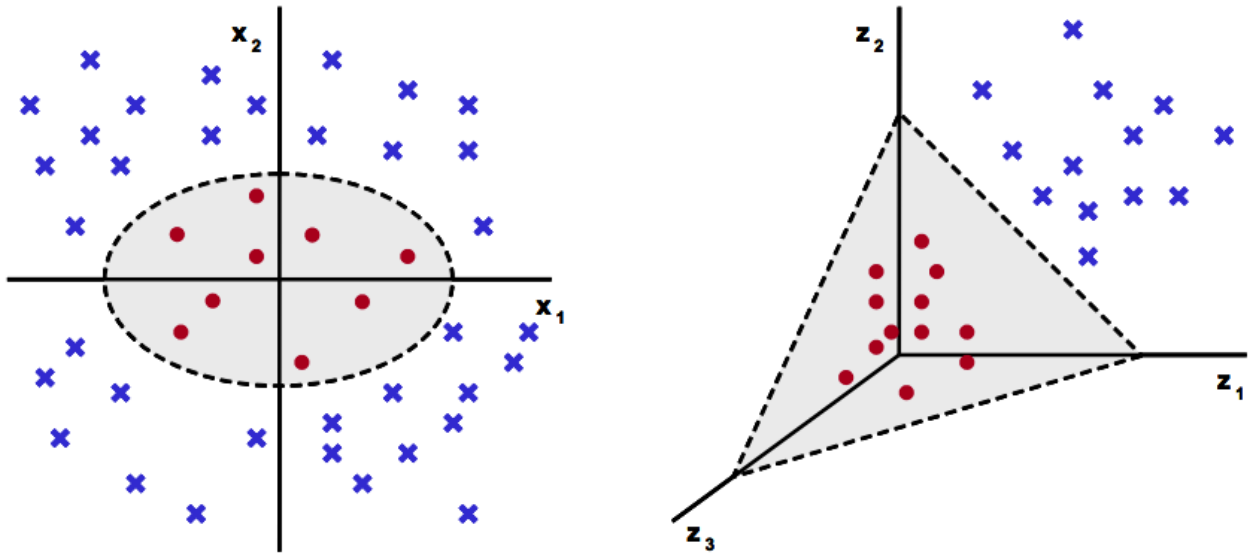
Sep 22

Feature Mapping

- Learn non-linear classifiers by mapping features

$$\varphi : \mathbb{R}^2 \to \mathbb{R}^3$$

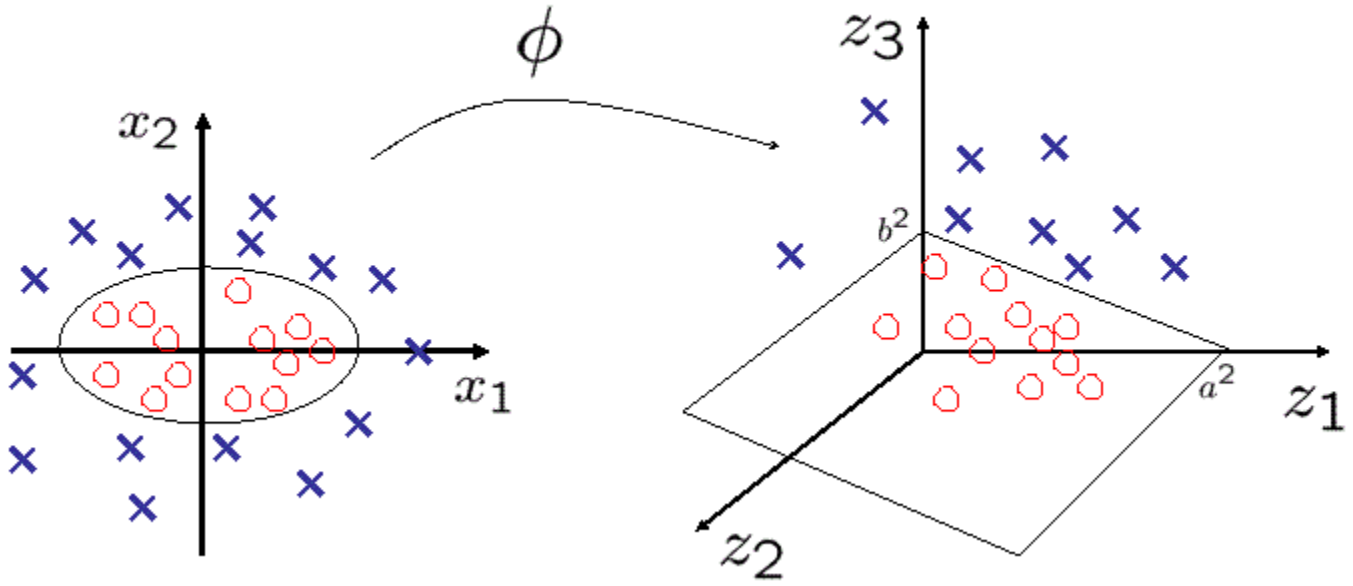
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



Can we learn the XOR function with this mapping?

Framework

Nonlinear model



build hypothesis $y = w^T \phi(x)$

Linear model

Quadratic Feature Map

- Let, $\mathbf{x} = [x_1, x_2, \ldots, x_D]$
- Then the quadratic feature map is defined as:

$$\begin{aligned}\phi(\mathbf{x}) = [&1, \sqrt{2}x_1, \sqrt{2}x_2, \ldots, \sqrt{2}x_D, \\ &x_1^2, x_1x_2, x_1x_3 \ldots, x_1x_D, \\ &x_2x_1, x_2^2, x_2x_3 \ldots, x_2x_D, \\ &\ldots, \\ &x_Dx_1, x_Dx_2, x_Dx_3 \ldots, x_D^2]\end{aligned}$$

- Contains all single and pairwise terms
- There are repetitions, e.g., x_1x_2 and x_2x_1, but hopefully the learning algorithm can handle redundant features

Drawbacks of Feature Mapping

- Computational
  - Suppose training time is linear in feature dimension, quadratic feature map squares the training time
- Memory
  - Quadratic feature map squares the memory required to store the training data
- Statistical
  - Quadratic feature mapping squares the number of parameters
  - For now lets assume that regularization will deal with overfitting

Quadratic Kernel

- The dot product between feature maps of **x** and **z** is:

$$\begin{aligned}
\phi(\mathbf{x})^T\phi(\mathbf{z}) &= 1 + 2x_1z_1 + 2x_2z_2, \ldots, 2x_Dz_D + x_1^2z_1^2 + x_1x_2z_1z_2 + \ldots + x_1x_Dz_1z_D + \ldots \\
&\quad \ldots + x_Dx_1z_Dz_1 + x_Dx_2z_Dz_2 + \ldots + x_D^2z_D^2 \\
&= 1 + 2\left(\sum_i x_iz_i\right) + \sum_{i,j} x_ix_jz_iz_j \\
&= 1 + 2\left(\mathbf{x}^T\mathbf{z}\right) + \left(\mathbf{x}^T\mathbf{z}\right)^2 \\
&= \left(1 + \mathbf{x}^T\mathbf{z}\right)^2 \\
&= K(\mathbf{x}, \mathbf{z})
\end{aligned}$$

- quadratic kernel

- Thus, we can compute $\emptyset(\mathbf{x})^T\emptyset(\mathbf{z})$ in almost the same time as needed to compute $\mathbf{x}^T\mathbf{z}$ (one extra addition and multiplication)
- We will rewrite various algorithms using only dot products (or kernel evaluations), and not explicit features

Perceptron Revisited

Input: training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$

feature map ϕ

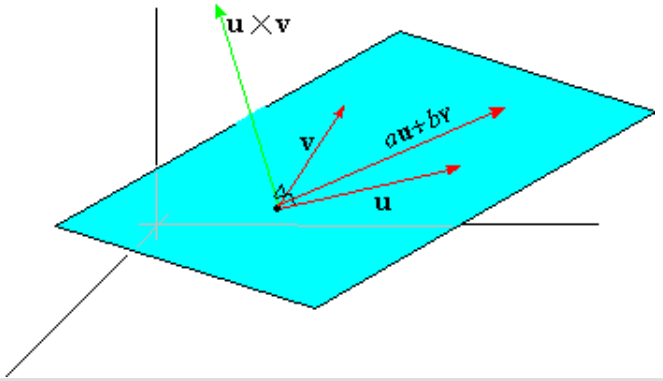
- Initialize $\mathbf{w} \leftarrow [0, \ldots, 0]$
- for iter = 1,…,T
  - for i = 1,..,n
    - predict according to the current model

$$\hat{y}_i = \begin{cases} +1 & \text{if } \mathbf{w}^T \phi(\mathbf{x}_i) > 0 \\ -1 & \text{otherwise} \end{cases}$$

    - if $y_i = \hat{y}_i$, no change
    - else, $\mathbf{w} \leftarrow \mathbf{w} + y_i \phi(\mathbf{x}_i)$

Properties of the Weight Vector

- Linear algebra recap:
  - Let U be set of vectors in R^D, i.e., $U = \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_D\}$ and $\mathbf{u}_i \in R^D$
  - Span(U) is the set of all vectors that can be represented as $\sum_i a_i \mathbf{u}_i$, such that $a_i \in R$
  - Null(U) is everything that is left i.e., $R^D \setminus$ Span(U)



Perceptron representer theorem: During the run of the perceptron training algorithm, the weight vector **w** is always in the span of $\phi(\mathbf{x}_1), \phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_D)$

$$\mathbf{w} = \sum_i \alpha_i \phi(\mathbf{x}_i) \qquad \text{updates} \qquad \alpha_i \leftarrow \alpha_i + y_i$$

$$\mathbf{w}^T \phi(\mathbf{z}) = \left(\sum_i \alpha_i \phi(\mathbf{x}_i)\right)^T \phi(\mathbf{z}) = \sum_i \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{z})$$

Kernelized Perception

Input: training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$

feature map ϕ

- Initialize $\alpha \leftarrow [0, 0, \ldots, 0]$
- for iter = 1,⋯,T
  - for i = 1,..,n
    - predict according to the current model

$$\hat{y}_i = \begin{cases} +1 & \text{if } \sum_n \alpha_n \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_i) > 0 \\ -1 & \text{otherwise} \end{cases}$$

    - if $y_i = \hat{y}_i$, no change
    - else, $\alpha_i = \alpha_i + y_i$

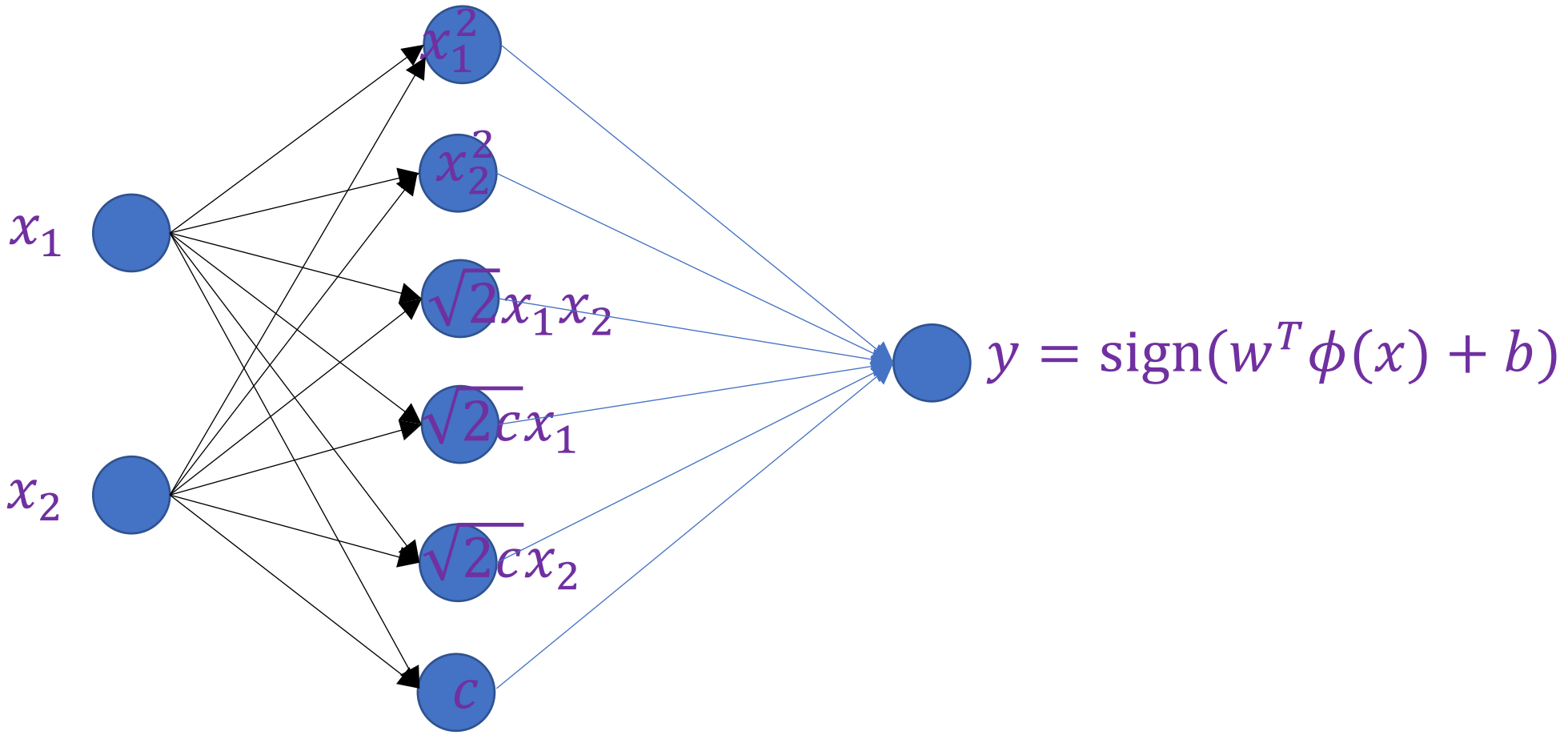
$$\phi(\mathbf{x})^T \phi(\mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^p$$

polynomial kernel of degree p

Polynomial Kernel

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^2, \quad K(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + c)^2 = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}\, x_1 x_2 \\ \sqrt{2c}\, x_1 \\ \sqrt{2c}\, x_2 \\ c \end{bmatrix} \cdot \begin{bmatrix} x_1'^2 \\ x_2'^2 \\ \sqrt{2}\, x_1' x_2' \\ \sqrt{2c}\, x_1' \\ \sqrt{2c}\, x_2' \\ c \end{bmatrix}$$

Polynomial kernel SVM as Neural Network



What Makes a Kernel?

- A kernel is a mapping $K: \mathcal{X} \times \mathcal{X} \rightarrow R$
- Functions that can be written as dot products are valid kernels

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

- Examples: polynomial kernel $K^d_{(\text{poly})}(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^d$
- Alternate characterization of a kernel
- A function $K: \mathcal{X} \times \mathcal{X} \rightarrow R$ is a kernel if K is positive semi-definite (psd)
- This property is also called as Mercer's condition
- This means that for all functions f that are squared integrable except the zero function, the following property holds:

$$\int \int f(\mathbf{x}) K(\mathbf{x}, \mathbf{z}) f(\mathbf{z}) d\mathbf{z} d\mathbf{x} > 0 \qquad \int f(\mathbf{x})^2 d\mathbf{x} < \infty$$

Why is this Characterization Useful?

- We can prove some properties about kernels that are otherwise hard to prove
- Theorem: If K_1 and K_2 are kernels, then $K_1 + K_2$ is also a kernel
- Proof:

$$\begin{aligned}\int\int f(\mathbf{x})K(\mathbf{x},\mathbf{z})f(\mathbf{z})d\mathbf{z}d\mathbf{x} &= \int\int f(\mathbf{x})\left(K_1(\mathbf{x},\mathbf{z})+K_2(\mathbf{x},\mathbf{z})\right)f(\mathbf{z})d\mathbf{z}d\mathbf{x} \\ &= \int\int f(\mathbf{x})K_1(\mathbf{x},\mathbf{z})f(\mathbf{z})d\mathbf{z}d\mathbf{x} + \int\int f(\mathbf{x})K_2(\mathbf{x},\mathbf{z})f(\mathbf{z})d\mathbf{z}d\mathbf{x} \\ &\geq 0+0\end{aligned}$$

- More generally if $K_1, K_2, \cdots, K_n$ are kernels then $\sum_i \alpha_i K_i$ with $\alpha_i \geqslant 0$, is a also a kernel
- Can build new kernels by linearly combining existing kernels

Why is this Characterization Useful?

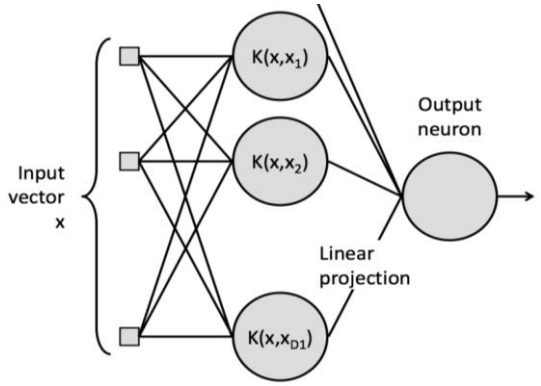
- We can show that the Gaussian function is a kernel
  - Also called as radial basis function (RBF) kernel

$$K_{(\mathrm{rbf})}(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma||\mathbf{x} - \mathbf{z}||^2\right)$$

- Lets look at the classification function using a SVM with RBF kernel:

$$f(\mathbf{z}) = \sum_i \alpha_i K_{(\mathrm{rbf})}(\mathbf{x}_i, \mathbf{z})$$

$$= \sum_i \alpha_i \exp\left(-\gamma||\mathbf{x}_i - \mathbf{z}||^2\right)$$



- This is similar to a two layer network with the RBF as the link function
- Gaussian kernels are examples of universal kernels — they can approximate any function in the limit as training data goes to infinity

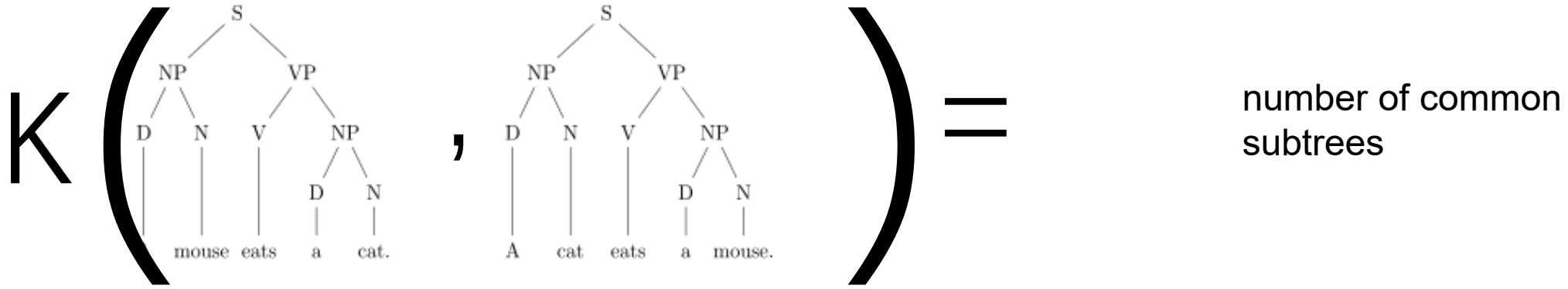
Kernels in Practice

- Feature mapping via kernels often improves performance
- MNIST digits test error:
  - **8.4%** SVM linear
  - **1.4%** SVM RBF
  - **1.1%** SVM polynomial (d=4)

60,000 training examples

Kernels over General Structures

- Kernels can be defined over any pair of inputs such as strings, trees and graphs!
- Kernel over trees:



  - This can be computed efficiently using dynamic programming
  - Can be used with SVMs, perceptrons, k-means, etc
- For strings number of common substrings is a kernel
- Graph kernels that measure graph similarity (e.g. number of common subgraphs)

have been used to predict toxicity of chemical structures

Graph Kernels

- Graph kernels compute the similarity between pairs of graphs, based on common substructures they share. A wide variety of substructures has been proposed , such as random walks, shortest paths, subtrees, graphlets, WL.

- Instance of R-convolution kernels by Haussler (1999)
- R-convolution kernels compare decompositions of two structured objects

$$k_{convolution}(x, x') = \sum_{(x_d, x) \in \mathrm{R}} \sum_{(x'_d, x') \in \mathrm{R}} k_{parts}(x_d, x'_d)$$

Random Walks

- Count common walks in two input graphs G and G'
- Walks are sequences of nodes that allow repetitions of nodes
- Walks of length K can be computed by looking at the k-th power of the adjacency matrix
- Construct direct product graph of G and G'

$$k_{\times}(G, G') = \sum_{i,j=1}^{|V_{\times}|} [\sum_{k=0}^{\infty} \lambda^k A_{\times}^k]_{ij}$$

Shortest Path

- Compute all-pairs-shortest-paths for G and G'
- Define a kernel by comparing all pairs of shortest path lengths from G and G'

$$k(G, G') = \sum_{v_i, v_j \in G} \sum_{v'_k, v'_l \in G'} k_{length}(d(v_i, v_j), d(v'_k, v'_l))$$

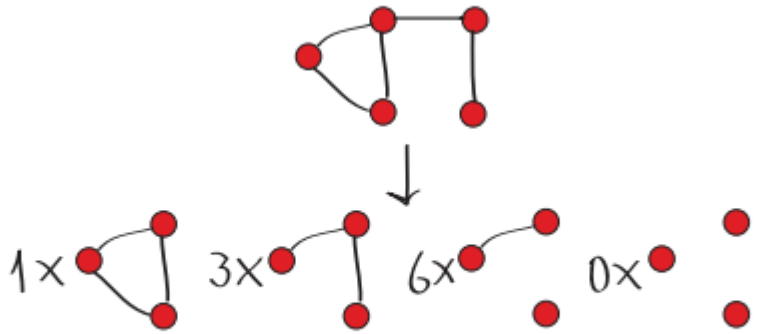
- $d(v_i, v_j)$ is the length of the shortest path between node $v_i \; and \; v_j$
- k_{length} is a kernel that compares the lengths of two shortest paths, for instance,

$$k(d(v_i, v_j), d(v'_k, v'_l)) = d(v_i, v_j) * d(v'_k, v'_l), \text{ or}$$

$$k(d(v_i, v_j), d(v'_k, v'_l)) = \begin{cases} 1 & \text{if } d(v_i, v_j) = d(v'_k, v'_l) \\ 0 & \text{otherwise} \end{cases}$$

Graphlet Kernels

- Graphlet kernel decomposes graphs into graphlets (i.e., small subgraphs with K nodes where k∈{3,4,5}) and counts matching graphlets in the input graphs.

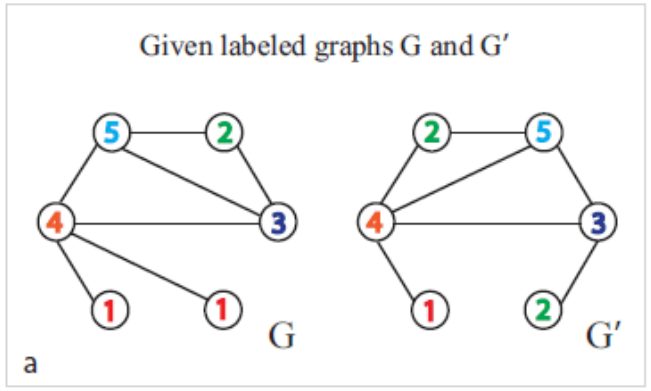


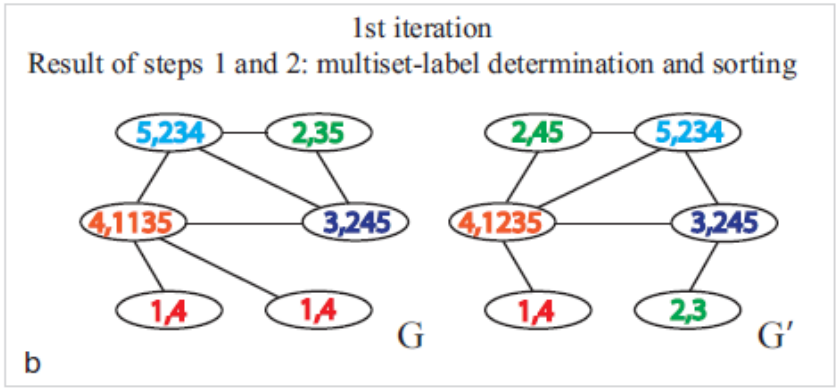
In this figure $n = 5$, $k = 3$, $f_G = (1, 3, 6, 0)$.

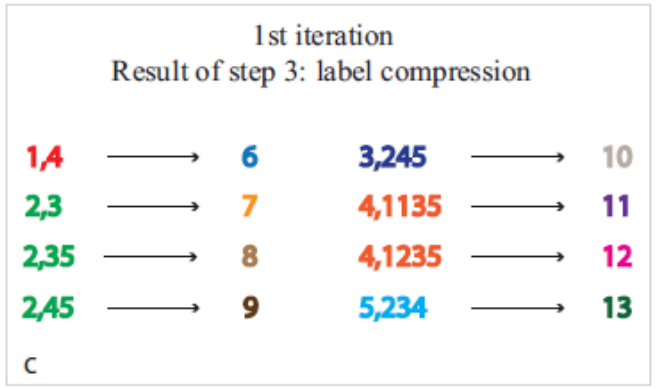
Graph Isomorphism

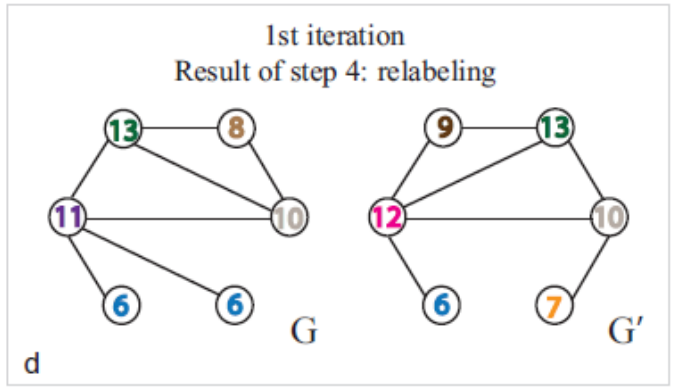
- Two graphs G and G' are said to b isomorphic, denoted by G ~ G', if there is a one-to-one correspondence, called an isomorphism, between the nodes of the graph such that two nodes are adjacent in G if and only if their corresponding nodes are adjacent in G'.
- Graph Isomorphism test is a hard problem.

WL Test









For each iteration:
- Step-1: neighborhood aggregation
- Step-2: label compression by hashing
- Step-3: relabeling

WL Test

The classical 1-WL test, keeps a state of each node that refines by aggregating their neighbors state. It outputs an embedding of the graph that corresponds to the state of every node. We say that the WL succeeds at distinguishing a pair of non-isomorphic graph G, G' if WL(G)≠ WL(G').

Algorithm *1-WL (color refinement)*
Input: $G = (V, E, X_V)$
1. $c_v^0 \leftarrow \text{hash}(X_v)$ for all $v \in V$
2. **repeat**
3. $\quad c_v^\ell \leftarrow \text{hash}(c_v^{\ell-1}, \{\!\{c_w^{\ell-1} : w \in \mathcal{N}_G(v)\}\!\})\ \forall v \in V$
4. **until** $(c_v^\ell)_{v\in V} = (c_v^{\ell-1})_{v\in V}$
5. **return** $\{\!\{c_v^\ell : v \in V\}\!\}$

WL vs GNN

Xu et al. (2019) and Morris et al. (2019) proved that,

$$GNN \leq WL$$

Xu et al. (2019) further proved, if the aggregation\readout functions are injective,

$$GNN = WL$$

Slides Credit

[1] Subhransu Maji. Kernels in CMPSCI689.
[2] Yingyu Liang. SVM II in COS495.
[3] Karsten et al. An Introduction to Graph Kernels.