**Non-linear Support Vector Machines**

**Non-linearly separable problems**

- Hard-margin SVM can address linearly separable problems
- Soft-margin SVM can address linearly separable problems with outliers
- Non-linearly separable problems need a higher expressive power (i.e. more complex feature combinations)
- We do not want to loose the advantages of linear separators (i.e. large margin, theoretical guarantees)

*Solution*

- Map input examples in a higher dimensional *feature space*
- Perform linear classification in this higher dimensional space

**Non-linear Support Vector Machines**

**feature map**

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

- $\Phi$ is a function mapping each example to a higher dimensional space $\mathcal{H}$
- Examples $\boldsymbol{x}$ are replaced with their feature mapping $\Phi(\boldsymbol{x})$
- The feature mapping should increase the expressive power of the representation (e.g. introducing features which are combinations of input features)
- Examples should be (approximately) linearly separable in the mapped space

**Feature map**

Homogeneous ($d = 2$) Inhomogeneous ($d = 2$)

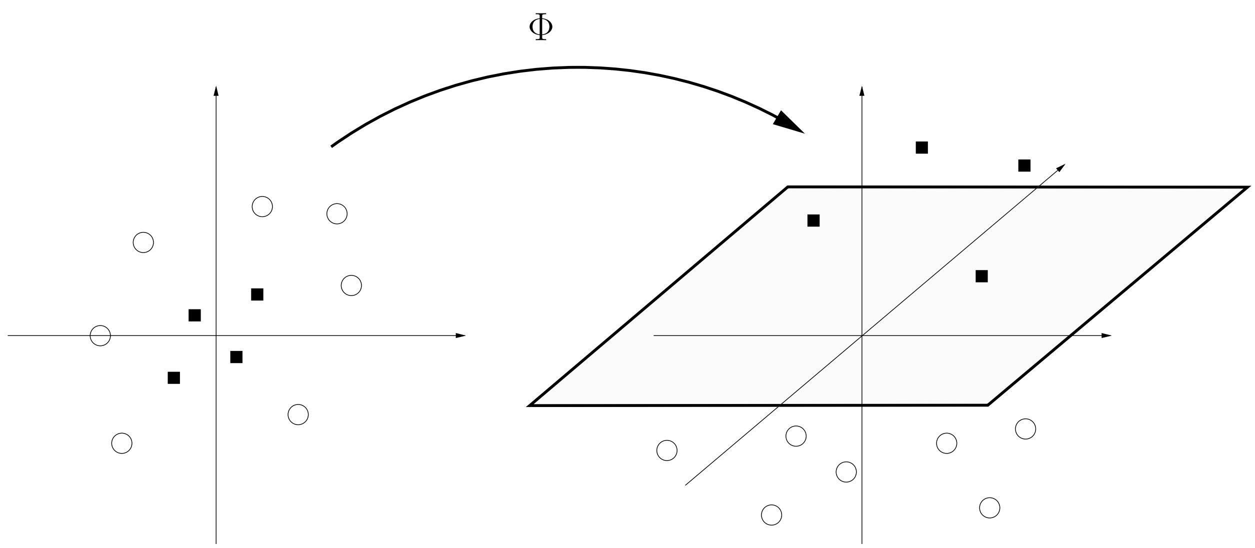
$$\Phi \left( \begin{array}{c} x_1 \\ x_2 \end{array} \right) = \left( \begin{array}{c} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{array} \right)$$

$$\Phi \left( \begin{array}{c} x_1 \\ x_2 \end{array} \right) = \left( \begin{array}{c} x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{array} \right)$$

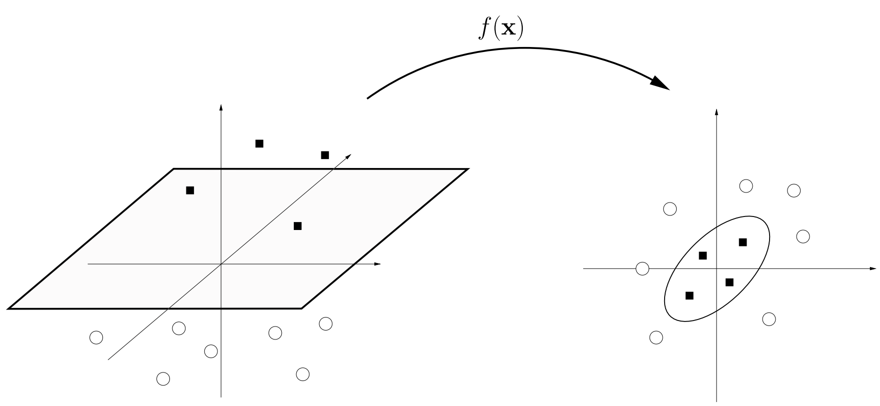
**Polynomial mapping**

- Maps features to all possible conjunctions (i.e. products) of features:
  1. *of a certain degree* $d$ (*homogeneous* mapping)
  2. *up to a certain degree* (*inhomogeneous* mapping)

**Feature map**




**Non-linear Support Vector Machines**




**Linear separation in feature space**

- SVM algorithm is applied just replacing $\boldsymbol{x}$ with $\Phi(\mathbf{x})$:

$$f(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + w_0$$

- A linear separation (i.e. hyperplane) in feature space corresponds to a non-linear separation in input space, e.g.:

$$f\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \text{sgn}(w_1 x_1^2 + w_2 x_1 x_2 + w_3 x_2^2 + w_0)$$

**Kernel Machines**

**Kernel trick**

- Feature mapping $\Phi(\cdot)$ can be very high dimensional (e.g. think of polynomial mapping)
- It can be highly expensive to explicitly compute it

- Feature mappings **appear only in dot products** in dual formulations
- The *kernel trick* consists in replacing these dot products with an equivalent kernel function:

$$k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T \Phi(\mathbf{x}')$$

- The kernel function uses examples in input (not feature) space

**Kernel trick**

**Support vector classification**

- Dual optimization problem

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in \mathbb{R}^m} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \underbrace{\Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)}_{k(\mathbf{x}_i, \mathbf{x}_j)} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \quad i = 1, \dots, m \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned}$$

- Dual decision function

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \underbrace{\Phi(\mathbf{x}_i)^T \Phi(\mathbf{x})}_{k(\mathbf{x}_i, \mathbf{x})}$$

**Kernel trick**

**Polynomial kernel**

- Homogeneous:

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^d$$

- E.g. ($d = 2$)

$$\begin{aligned} k\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix}\right) &= (x_1 x'_1 + x_2 x'_2)^2 \\ &= (x_1 x'_1)^2 + (x_2 x'_2)^2 + 2 x_1 x'_1 x_2 x'_2 \\ &= \underbrace{\begin{pmatrix} x_1^2 & \sqrt{2} x_1 x_2 & x_2^2 \end{pmatrix}}_{\Phi(\mathbf{x})^T} \underbrace{\begin{pmatrix} x_1'^2 \\ \sqrt{2} x'_1 x'_2 \\ x_2'^2 \end{pmatrix}}_{\Phi(\mathbf{x}')} \end{aligned}$$

**Kernel trick**
**Polynomial kernel**

- Inhomogeneous: $$k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T\mathbf{x}')^d$$

- E.g. ($d = 2$)

$$\begin{aligned}
k(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}) &= (1 + x_1x_1' + x_2x_2')^2 \\
&= 1 + (x_1x_1')^2 + (x_2x_2')^2 + 2x_1x_1' + 2x_2x_2' + 2x_1x_1'x_2x_2' \\
&= \underbrace{\begin{pmatrix} 1 & \sqrt{2}x_1 & \sqrt{2}x_2 & x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{pmatrix}}_{\Phi(\mathbf{x})^T} \underbrace{\begin{pmatrix} 1 \\ \sqrt{2}x_1' \\ \sqrt{2}x_2' \\ x_1'^2 \\ \sqrt{2}x_1'x_2' \\ x_2'^2 \end{pmatrix}}_{\Phi(\mathbf{x}')}
\end{aligned}$$

**Valid Kernels**

**Dot product in feature space**

- A valid kernel is a (similarity) function defined in cartesian product of input space:

$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$

- corresponding to a dot product in a (certain) feature space:

$$k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T\Phi(\mathbf{x}')$$

*Note*

- The kernel generalizes the notion of dot product to arbitrary input space (e.g. protein sequences)
- It can be seen as a measure of similarity between objects

**Valid Kernels**

**Gram matrix**

- Given examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ and kernel function $k$
- The *Gram matrix* $K$ is the (symmetric) matrix of pairwise kernels between examples:

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \quad \forall i, j$$

**Valid Kernels**

**Positive definite matrix**

- A symmetric $m \times m$ matrix $K$ is *positive definite* (p.d.) if

$$\sum_{i,j=1}^{m} c_i c_j K_{ij} \geq 0, \quad \forall \mathbf{c} \in \mathbb{R}^m$$

If equality only holds for $\mathbf{c} = \mathbf{0}$, the matrix is *strictly positive definite* (s.p.d)

*Alternative conditions*

- All eigenvalues are non-negative (positive for s.p.d.)
- There exists a matrix $B$ such that

$$K = B^T B$$

**Valid Kernels**

**Positive definite kernels**

- A positive definite kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ giving rise to a p.d. Gram matrix for any $m$ and $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$
- Positive definiteness is necessary and sufficient condition for a kernel to correspond to a dot product of *some* feature map $\Phi$

*How to verify kernel validity*

- Prove its positive definiteness (difficult)
- Find out a corresponding feature map (see polynomial example)
- Use kernel combination properties (we'll see)

**Kernel machines**
**Support vector regression**

- Dual problem:

$$\begin{aligned}
\max_{\boldsymbol{\alpha} \in \mathbb{R}^m} \quad & -\frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) \underbrace{\Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)}_{k(\mathbf{x}_i, \mathbf{x}_j)} \\
& -\epsilon \sum_{i=1}^{m} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} y_i (\alpha_i^* - \alpha_i) \\
\text{subject to} \quad & \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) = 0 \quad \alpha_i, \alpha_i^* \in [0, C] \quad \forall i \in [1, m]
\end{aligned}$$

- Regression function:

$$f(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + w_0 = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) \underbrace{\Phi(\mathbf{x}_i)^T \Phi(\mathbf{x})}_{k(\mathbf{x}_i, \mathbf{x})} + w_0$$

**Kernel machines**

**(Stochastic) Perceptron:** $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

1. Initialize $\mathbf{w} = \mathbf{0}$
2. Iterate until all examples correctly classified:
   (a) For each incorrectly classified training example $(\boldsymbol{x}_i, y_i)$:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$$

**Kernel Perceptron:** $f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x})$

1. Initialize $\alpha_i = 0 \; \forall i$
2. Iterate until all examples correctly classified:
   (a) For each incorrectly classified training example $(\boldsymbol{x}_i, y_i)$:

$$\alpha_i \leftarrow \alpha_i + \eta y_i$$

**Kernels**

**Basic kernels**

- linear kernel:

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$$

- polynomial kernel:

$$k_{d,c}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + c)^d$$

**Kernels**

**Gaussian kernel**

$$k_\sigma(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right) = \exp\left(-\frac{\mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{x}' + \mathbf{x}'^T\mathbf{x}'}{2\sigma^2}\right)$$

- Depends on a *width* parameter $\sigma$
- The smaller the width, the more prediction on a point only depends on its nearest neighbours
- Example of *Universal* kernel: they can uniformly approximate any arbitrary continuous target function (pb of number of training examples and choice of $\sigma$)

## Kernels
### Kernels on structured data

- Kernels are generalization of dot products to arbitrary domains
- It is possible to design kernels over structured objects like sequences, trees or graphs
- The idea is designing a pairwise function measuring the similarity of two objects
- This measure has to satisfy the p.d. conditions to be a valid kernel

*Match (or delta) kernel*

$$k_\delta(x, x') = \delta(x, x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise.} \end{cases}$$

- Simplest kernel on structures
- $x$ does not need to be a vector! (no boldface to stress it)

### Kernels on sequences

$$\texttt{x = ABAABA} \qquad \texttt{x' = AAABB}$$

$$\Phi(x) \downarrow \qquad\qquad \downarrow \Phi(x')$$

$$\begin{array}{l} \text{AAA} \\ \text{AAB} \\ \text{ABA} \\ \text{ABB} \\ \text{BAA} \\ \text{BAB} \\ \text{BBA} \\ \text{BBB} \end{array} \quad \begin{pmatrix} 0 \\ 1 \\ 2 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad k(x, x') = 1$$

### Spectrum kernel

- Feature space is space of all possible k-grams (subsequences)
- An efficient procedure based on suffix trees allows to compute kernel without explicitly building feature maps

## Kernels
### Kernel combination

- Simpler kernels can combined using certain operators (e.g. sum, product)
- Kernel combination allows to design complex kernels on structures from simpler ones
- Correctly using combination operators guarantees that complex kernels are p.d.

*Note*

- Simplest constructive approach to build valid kernels

**Kernel combination**

**Kernel Sum**

- The sum of two kernels corresponds to the *concatenation* of their respective feature spaces:

$$
\begin{aligned}
(k_1 + k_2)(x, x') &= k_1(x, x') + k_2(x, x') \\
&= \Phi_1(x)^T \Phi_1(x') + \Phi_2(x)^T \Phi_2(x') \\
&= (\Phi_1(x)\ \Phi_2(x)) \begin{pmatrix} \Phi_1(x') \\ \Phi_2(x') \end{pmatrix}
\end{aligned}
$$

- The two kernels can be defined on **different** spaces (*direct* sum, e.g. string spectrum kernel plus string length)

**Kernel combination**
**Kernel Product**

- The product of two kernels corresponds to the Cartesian products of their features:

$$
\begin{aligned}
(k_1 \times k_2)(x, x') &= k_1(x, x') k_2(x, x') \\
&= \sum_{i=1}^{n} \Phi_{1i}(x) \Phi_{1i}(x') \sum_{j=1}^{m} \Phi_{2j}(x) \Phi_{2j}(x') \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} (\Phi_{1i}(x) \Phi_{2j}(x))(\Phi_{1i}(x') \Phi_{2j}(x')) \\
&= \sum_{k=1}^{nm} \Phi_{12k}(x) \Phi_{12k}(x') = \Phi_{12}(x)^T \Phi_{12}(x')
\end{aligned}
$$

- where $\Phi_{12}(x) = \Phi_1(x) \times \Phi_2(x)$ is the Cartesian product
- the product can be between kernels in different spaces (*tensor* product)

**Kernel combination**

**Linear combination**

- A kernel can be rescaled by an arbitrary positive constant: $k_\beta(x, x') = \beta k(x, x')$
- We can e.g. define linear combinations of kernels (each rescaled by the desired weight):

$$
k_{sum}(x, x') = \sum_{k=1}^{K} \beta_k k_k(x, x')
$$

*Note*

- The weights of the linear combination can be learned simultaneously to the predictor weights (the alphas)
- This amounts at performing *kernel learning*

**Kernel combination**

**Kernel normalization**

- Kernel values can often be influenced by the dimension of objects
- E.g. a longer string has more substrings $\rightarrow$ higher kernel value
- This effect can be reduced *normalizing* the kernel

*Cosine normalization*

- Cosine normalization computes the cosine of the dot product in feature space:

$$\hat{k}(x, x') = \frac{k(x, x')}{\sqrt{k(x, x)k(x', x')}}$$

**Kernel combination**

**Kernel composition**

- Given a kernel over structured data $k(x, x')$
- it is always possible to use a basic kernel on top of it, e.g.:

$$\begin{aligned} (k_{d,c} \circ k))(x, x') &= (k(x, x') + c)^d \\ (k_\sigma \circ k)(x, x') &= \exp\left(-\frac{k(x, x) - 2k(x, x') + k(x', x')}{2\sigma^2}\right) \end{aligned}$$

- it corresponds to the **composition** of the mappings associated with the two kernels
- E.g. all possible conjunctions of up to $d$ k-grams for string kernels

**Kernels on graphs**

**Weistfeiler-Lehman graph kernel**

- Efficient graph kernel for large graphs
- Relies on (approximation of) Weistfeiler-Lehman test of graph isomorphism
- Defines a family of graph kernels

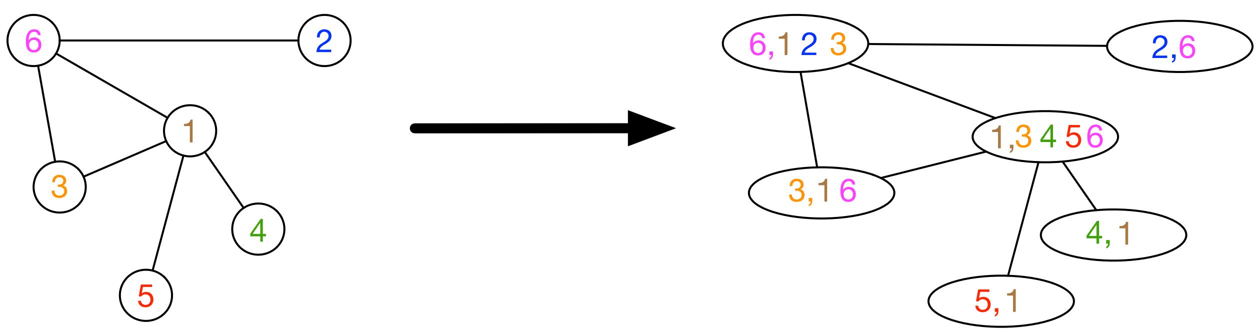
## Kernels on graphs

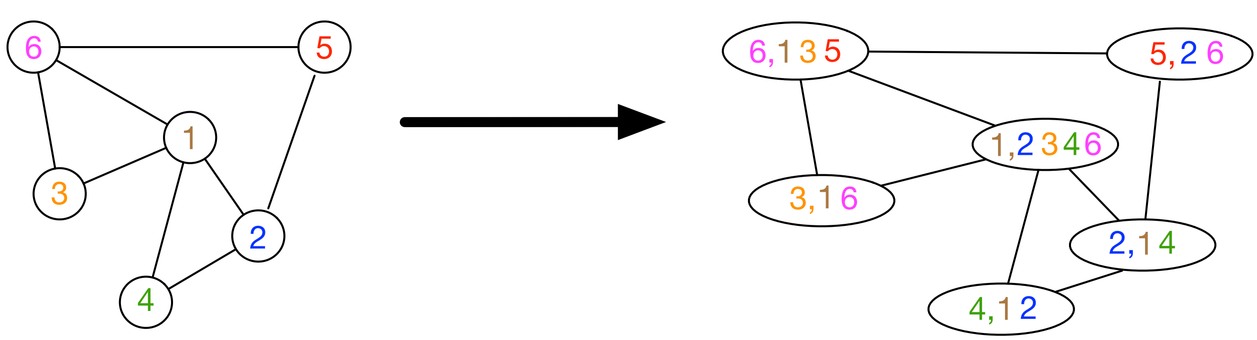
### Weistfeiler-Lehman (WL) isomorphism test

Given $G = (\mathcal{V}, \mathcal{E})$ and $G' = (\mathcal{V}', \mathcal{E}')$, with $n = |\mathcal{V}| = |\mathcal{V}'|$. Let $L(G) = \{l(v)|v \in \mathcal{V}\}$ be the set of labels in $G$, and let $L(G) == L(G')$. Let $label(s)$ be a function assigning a unique label to a string.

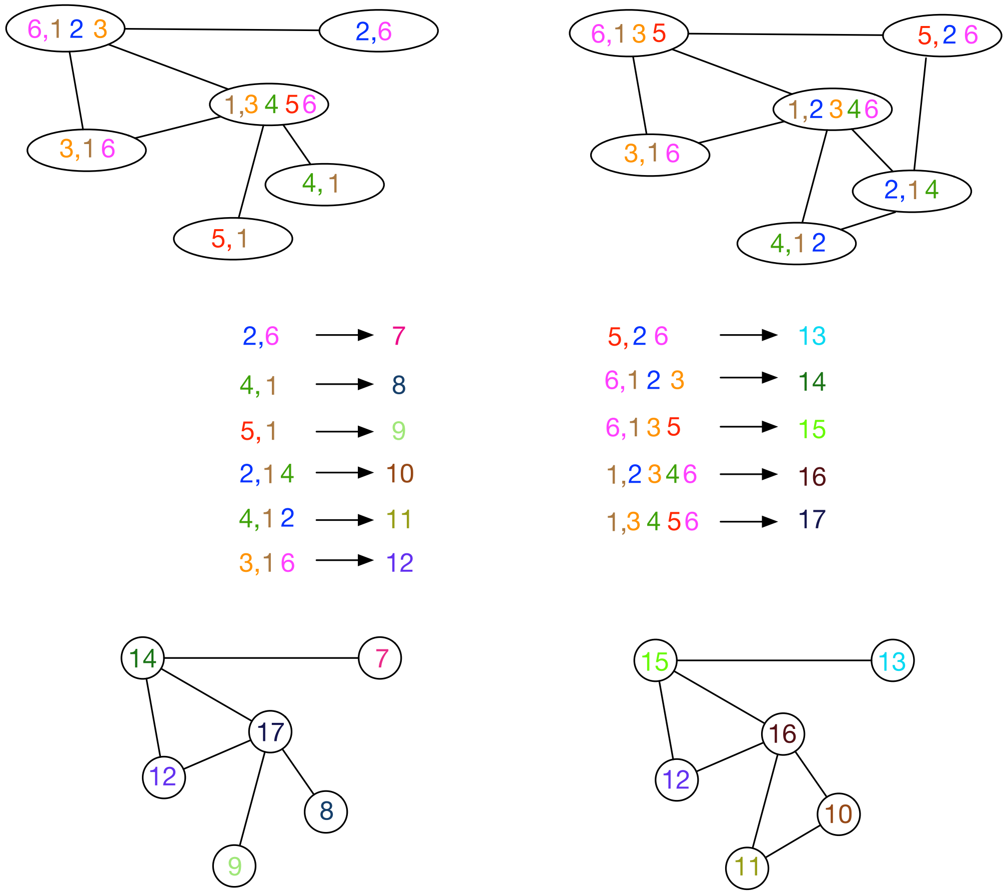
- Set $l_0(v) = l(v)$ for all $v$.
- For $i \in [1, n-1]$
  1. For each node $v$ in $G$ and $G'$
  2. Let $M_i(v) = \{l_{i-1}(u)|u \in neigh(v)\}$
  3. Concatenate the sorted labels of $M_i(v)$ into $s_i(v)$
  4. Let $l_i(v) = label(l_{i-1}(v) \circ s_i(v))$ ($\circ$ is concatenation)
  5. If $L_i(G) \neq L_i(G')$
  6. Return **Fail**
- Return **Pass**

### WL isomorphism test: string determination



**WL isomorphism test: relabeling**



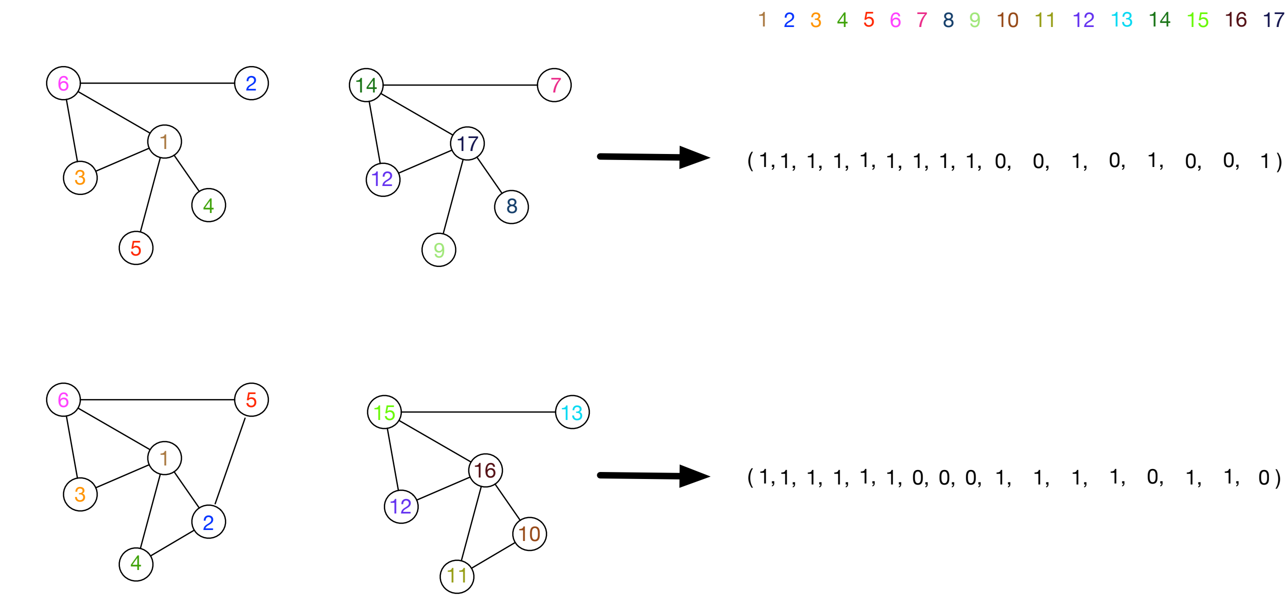

**Kernels on graphs**

**Weistfeiler-Lehman graph kernel**

- Let $\{G_0, G_1, \ldots, G_h\} = \{(\mathcal{V}, \mathcal{E}, l_0), (\mathcal{V}, \mathcal{E}, l_1), \ldots, (\mathcal{V}, \mathcal{E}, l_h)\}$ be a sequence of graphs made from $G$, where $l_i$ is the node labeling of the i-th WL iteration.
- Let $k : G \times G' \to \mathbb{R}$ be any kernel on graphs.
- The Weistfeiler-Lehman graph kernel is defined as:

$$k^h_{WL}(G, G') = \sum_{i=0}^{h} k(G_i, G'_i)$$

**Example: WL subtree kernel**




## References


**kernel trick** C. Burges, *A tutorial on support vector machines for pattern recognition*, Data Mining and Knowledge Discovery, 2(2), 121-167, 1998.

**kernel properties** J.Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004 (Section 3)

**kernels** J.Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004 (Section 9)

**graph kernels** N. Shervashidze, P. Schweitzer, E. Jan van Leeuwen, K. Mehlhorn, and K. Borgwardt. *Weisfeiler-Lehman Graph Kernels*. J. Mach. Learn. Res., 2011.