Universität Potsdam
Institut für Informatik
Lehrstuhl Maschinelles Lernen

Kernel Methods

Tobias Scheffer

Contents

- Feature mappings
  - Representer Theorem
- Kernel learning algorithms
  - Kernel ridge regression
  - Kernel perceptron,
  - Dual SVM
- Mercer map
- Kernel functions
  - Polynomial, RBF
  - For time series, strings, graphs

Review: Linear Models

- Linear models: $f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\boldsymbol{\theta}$

- Regularized empirical risk minimization:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i) + \lambda\Omega(\boldsymbol{\theta})$$

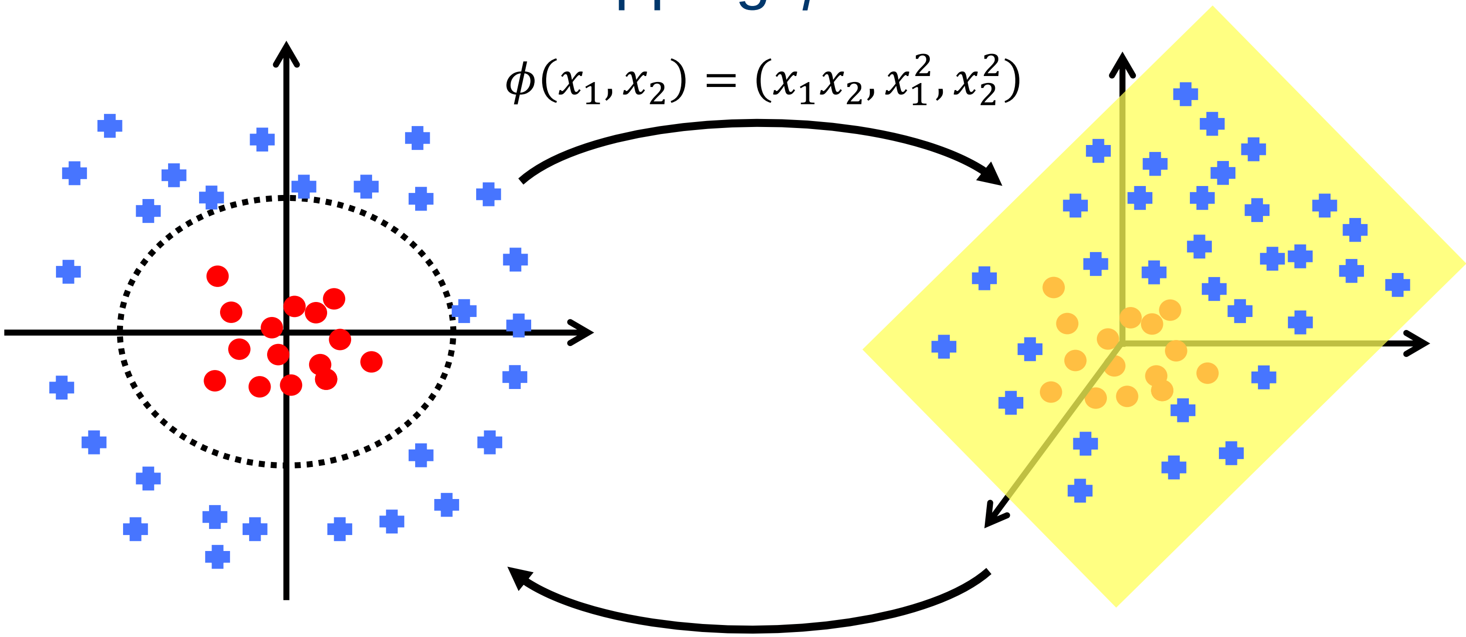
- Choice of loss & regularizer gives different methods
  - Perceptron, SVM, ridge regression, …

Feature Mappings

- Models constained to hyperplane in feature space: $H_{\boldsymbol{\theta}} = \{\mathbf{x} | \mathbf{x}^{\mathrm{T}} \boldsymbol{\theta} = 0\}$.
- Use mapping ϕ to embed instances $\mathbf{x} \in X$ in higher-dimensional feature space.
- Find hyperplane in higher-dimensional space, corresponds to non-linear surface in feature space.
- Kernel trick: Feature space $\phi(X)$ need not be represented explicitly, can be infinite-dimensional.

Feature Mappings

- All linear methods can be made non-linear by means of feature mapping ϕ.



- Hyperplane in feature space corresponds to a nonlinear surface in original space.

Feature Mappings

- Instances:

$$\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix}$$

- Feature Mapping:

$$\mathbf{\Phi} = \begin{pmatrix} \phi(\mathbf{x}_1)^{\mathrm{T}} \\ \vdots \\ \phi(\mathbf{x}_n)^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} \phi(\mathbf{x}_1)_1 & \cdots & \phi(\mathbf{x}_1)_{m'} \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_n)_1 & \cdots & \phi(\mathbf{x}_n)_{m'} \end{pmatrix}$$

Feature Mappings

- Feature mapping $\phi(\mathbf{x})$ can be high dimensional.
  - The size of estimated parameter vector $\boldsymbol{\theta}$ depends on the dimensionality of ϕ – could be infinite!
- Computation of $\phi(\mathbf{x})$ can be expensive.
  - ϕ must be computed for each training point $\mathbf{x}_i$ & for each prediction x.
- How can we adapt linear methods to efficiently incorporate high dimensional ϕ?

Representer Theorem: Observation

- Perceptron algorithm:

$$\begin{array}{ll} \texttt{IF} & y_i f_{\boldsymbol{\theta}}(\mathbf{x}_i) \leq 0 \\ \texttt{THEN} & \boldsymbol{\theta} = \boldsymbol{\theta} + y_i \mathbf{x}_i \end{array}$$

- Resulting parameter vector is a linear combination of instances: $\boldsymbol{\theta}^* = \sum_{i=1}^{n} \alpha_i \, y_i \mathbf{x}_i$
- Sufficient to determine coefficients α_i, independent of dimensionality of feature space.
- Underlying general principle?

Contents

- Feature mappings
  - Representer Theorem
- Kernel learning algorithms
  - Kernel ridge regression
  - Kernel perceptron,
  - Dual SVM
- Mercer map
- Kernel functions
  - Polynomial, RBF
  - For time series, strings, graphs

Representer Theorem

Theorem: If $g(\circ)$ is strictly monotonically increasing, then the $\boldsymbol{\theta}^*$ that minimizes

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ell(\boldsymbol{\theta}^T \phi(\mathbf{x}_i), y_i) + g(\|f_{\boldsymbol{\theta}}\|_2)$$

has the form $\boldsymbol{\theta}^* = \sum_{i=1}^{n} \alpha_i^* \phi(\mathbf{x}_i)$, with $\alpha_i^* \in \mathbb{R}$.

$$f_{\boldsymbol{\theta}^*}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x})$$

Inner product is a measure for similarity between instances

Generally $\boldsymbol{\theta}^*$ is any vector in Φ, but we show it must be in the span of the data.

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i) + g(\|f_{\boldsymbol{\theta}}\|_2)$$

Representer Theorem: Proof (Sketch)

- Orthogonal Decomposition:
  - $\boldsymbol{\theta}^* = \boldsymbol{\theta}_{\parallel} + \boldsymbol{\theta}_{\perp}$, with $\boldsymbol{\theta}_{\parallel} \in \Theta_{\parallel} = \{\sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i) \,|\alpha_i \in \mathbb{R}\}$ and $\boldsymbol{\theta}_{\perp} \in \Theta_{\perp} = \{\boldsymbol{\theta} \in \Theta \big| \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}_{\parallel} = 0 \;\forall\; \boldsymbol{\theta}_{\parallel} \in \Theta_{\parallel}\}$

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i) + g(\|f_{\boldsymbol{\theta}}\|_2)$$

Representer Theorem: Proof (Sketch)

- Orthogonal Decomposition:
  - $\boldsymbol{\theta}^* = \boldsymbol{\theta}_\parallel + \boldsymbol{\theta}_\perp$, with $\boldsymbol{\theta}_\parallel \in \Theta_\parallel = \{\sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i) \,|\alpha_i \in \mathbb{R}\}$ and $\boldsymbol{\theta}_\perp \in \Theta_\perp = \{\boldsymbol{\theta} \in \Theta \big| \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}_\parallel = 0 \;\forall\; \boldsymbol{\theta}_\parallel \in \Theta_\parallel\}$
- For any training point $\mathbf{x}_i$ it follows that

$$f_{\boldsymbol{\theta}^*}(\mathbf{x}_i) = {\boldsymbol{\theta}_\parallel}^{\mathrm{T}} \phi(\mathbf{x}_i) + {\boldsymbol{\theta}_\perp}^{\mathrm{T}} \phi(\mathbf{x}_i) = {\boldsymbol{\theta}_\parallel}^{\mathrm{T}} \phi(\mathbf{x}_i)$$

  - Why is ${\boldsymbol{\theta}_\perp}^{\mathrm{T}} \phi(\mathbf{x}_i) = 0$?

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i) + g(\|f_{\boldsymbol{\theta}}\|_2)$$

Representer Theorem: Proof (Sketch)

- Orthogonal Decomposition:
  - $\boldsymbol{\theta}^* = \boldsymbol{\theta}_\parallel + \boldsymbol{\theta}_\perp$, with $\boldsymbol{\theta}_\parallel \in \Theta_\parallel = \{\sum_{i=1}^n \alpha_i \phi(\mathbf{x}_i) \,|\alpha_i \in \mathbb{R}\}$ and $\boldsymbol{\theta}_\perp \in \Theta_\perp = \{\boldsymbol{\theta} \in \Theta \big| \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}_\parallel = 0 \;\; \forall \;\; \boldsymbol{\theta}_\parallel \in \Theta_\parallel\}$
- For any training point $\mathbf{x}_i$ it follows that
  $$f_{\boldsymbol{\theta}^*}(\mathbf{x}_i) = {\boldsymbol{\theta}_\parallel}^{\mathrm{T}} \phi(\mathbf{x}_i) + {\boldsymbol{\theta}_\perp}^{\mathrm{T}} \phi(\mathbf{x}_i) = {\boldsymbol{\theta}_\parallel}^{\mathrm{T}} \phi(\mathbf{x}_i)$$
  - $\sum_{i=1}^n \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)$ is independent of $\boldsymbol{\theta}_\perp$.
  - because ${\boldsymbol{\theta}_\perp}^{\mathrm{T}} \phi(\mathbf{x}_i) = 0$
- Finally from $g(\|\boldsymbol{\theta}^*\|_2) \geq g\left(\left\|\boldsymbol{\theta}_\parallel\right\|_2\right)$, it follows $\boldsymbol{\theta}_\perp = \mathbf{0}$.

$$g(\|\boldsymbol{\theta}^*\|_2) = g\left(\left\|\boldsymbol{\theta}_\parallel + \boldsymbol{\theta}_\perp\right\|_2\right) = g\left(\sqrt{\left\|\boldsymbol{\theta}_\parallel\right\|_2^2 + \|\boldsymbol{\theta}_\perp\|_2^2}\right) \geq g\left(\left\|\boldsymbol{\theta}_\parallel\right\|_2\right)$$

Since $\boldsymbol{\theta}_\perp^{\mathrm{T}} \boldsymbol{\theta}_\parallel = 0$ (Pythagoras‘ Theorem)

Since g is strictly monotonically increasing.

Representer Theorem

- The hyperplane $\boldsymbol{\theta}^*$, which minimizes
  - $L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ell\left(\boldsymbol{\theta}^{\mathrm{T}}\phi(\mathbf{x}_i), y_i\right) + \Omega(\boldsymbol{\theta})$
- can be represented as

$$f_{\boldsymbol{\theta}^*}(\mathbf{x}) = \boldsymbol{\theta}^{*\mathrm{T}}\phi(\mathbf{x}) = f_{\boldsymbol{\alpha}^*}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* \phi(\mathbf{x}_i)^{\mathrm{T}}\phi(\mathbf{x})$$

Primal vs. Dual View

- Primal decision function:

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}} \phi(\mathbf{x})$$

- Dual decision function:

$$f_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x})$$

Primal vs. Dual View

- Primal decision function:

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}} \phi(\mathbf{x})$$

- Dual decision function:

$$f_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x}) = \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{\Phi} \phi(\mathbf{x})$$

- Illustration:

$$\sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)^{\mathrm{T}}$$

$$= \begin{pmatrix} \alpha_1 & \dots & \alpha_n \end{pmatrix} \begin{pmatrix} - & \phi(\mathbf{x}_1)^{\mathrm{T}} & - \\ & \vdots & \\ - & \phi(\mathbf{x}_n)^{\mathrm{T}} & - \end{pmatrix} = \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{\Phi}$$

Primal vs. Dual View

- Primal decision function:
$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}}\phi(\mathbf{x})$$
- Dual decision function:
$$f_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)^{\mathrm{T}}\phi(\mathbf{x}) = \boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\phi(\mathbf{x})$$
- Duality between parameters:
$$\boldsymbol{\theta} = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$$
- Illustration:
$$\boldsymbol{\theta} = \begin{pmatrix} | & & | \\ \phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_n) \\ | & & | \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$$

Primal vs. Dual View

- Primal decision function:

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}} \phi(\mathbf{x})$$

- Dual decision function:

$$f_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum\nolimits_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x}) = \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{\Phi} \phi(\mathbf{x})$$

- Duality between parameters:

$$\boldsymbol{\theta} = \sum\nolimits_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i) = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

Primal vs. Dual View

- Primal view: $f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}}\phi(\mathbf{x})$
  - Model $\boldsymbol{\theta}$ has as many parameters as the dimensionality of $\phi(\mathbf{x})$.
  - Good if there are many examples with few attributes.

- Dual view: $f_{\boldsymbol{\alpha}}(\mathbf{x}) = \boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\phi(\mathbf{x})$
  - Model $\boldsymbol{\alpha}$ has as many parameters as there are examples.
  - Good if there are few examples with many attributes.
  - The representation $\phi(\mathbf{x})$ can even be infinite dimensional, as long as the inner product can be computed efficiently.

Kernel Functions

- Dual view of the decision function:

$$
\begin{aligned}
f_{\boldsymbol{\alpha}}(\mathbf{x}) &= \left(\sum_{i=1}^{n} \alpha_i \, \phi(\mathbf{x}_i)^{\mathrm{T}}\right) \phi(\mathbf{x}) \\
&= \sum_{i=1}^{n} \alpha_i \, \left(\phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x})\right) \\
&= \sum_{i=1}^{n} \alpha_i \, k(\mathbf{x}_i, \mathbf{x})
\end{aligned}
$$

- Where kernel function $k(\mathbf{x}_i, \mathbf{x})$ calculates the inner product $\phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x})$.

Kernel Functions

- Kernel functions can be understood as a measure of similarity between instances.
- Primal view on data: "what does $\mathbf{x}$ look like?"

$$\phi(\mathbf{x}) = \begin{pmatrix} \phi(x)_1 \\ \vdots \\ \phi(x)_{m'} \end{pmatrix} \Rightarrow \text{multiply by } \boldsymbol{\theta}^{\mathrm{T}}.$$

- Dual view on data: "how similar is $\mathbf{x}$ to each training instance?"

$$\boldsymbol{\Phi}\boldsymbol{\phi}(\mathbf{x}) = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}) \\ \vdots \\ k(\mathbf{x}_n, \mathbf{x}) \end{pmatrix} \Rightarrow \text{ multiply by } \boldsymbol{\alpha}^{\mathrm{T}}.$$

Kernel Functions

- Kernel function can be defined for
  - Vectors (linear, polynomial, RBF, …)
  - Strings
  - Images
  - Sequences, graphs
  - …
- Any kernel learning method can be applied to any type of data using a kernel for that type of data.

Contents

- Feature mappings
  - Representer Theorem
- Kernel learning algorithms
  - Kernel ridge regression
  - Kernel perceptron,
  - Dual SVM
- Mercer map
- Kernel functions
  - Polynomial, RBF
  - For time series, strings, graphs

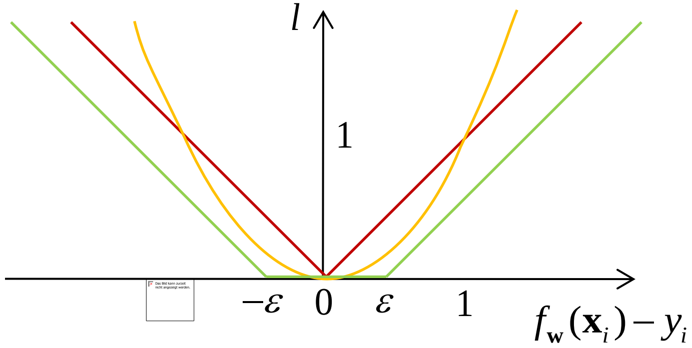
Kernel Ridge Regression

- Squared loss:

$$\ell_2(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i) = (f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2$$

- L2 regularization:

$$\Omega_2(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$$



Kernel Ridge Regression

- Minimize

$$\boldsymbol{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(\boldsymbol{\theta}^{\mathrm{T}} \phi(\mathbf{x}) - y_i\right)^2 + \lambda \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}$$

Kernel Ridge Regression

- Minimize

$$\boldsymbol{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(\boldsymbol{\theta}^{\mathrm{T}} \phi(\mathbf{x}) - y_i\right)^2 + \lambda \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}$$
$$= (\boldsymbol{\Phi}\boldsymbol{\theta} - \mathbf{y})^{\mathrm{T}} (\boldsymbol{\Phi}\boldsymbol{\theta} - \mathbf{y}) + \lambda \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}$$

- Why?

$$(\boldsymbol{\Phi}\boldsymbol{\theta} - \mathbf{y}) = \begin{pmatrix} - & \phi(\mathbf{x}_1)^{\mathrm{T}} & - \\ & \vdots & \\ - & \phi(\mathbf{x}_n)^{\mathrm{T}} & - \end{pmatrix} \begin{pmatrix} \boldsymbol{\theta}_1 \\ \vdots \\ \boldsymbol{\theta}_m \end{pmatrix} - \mathbf{y}$$
$$= \begin{pmatrix} \phi(\mathbf{x}_1)^{\mathrm{T}} \boldsymbol{\theta} - y_1 \\ \vdots \\ \phi(\mathbf{x}_n)^{\mathrm{T}} \boldsymbol{\theta} - y_n \end{pmatrix}$$

Kernel Ridge Regression

- Minimize
$$\boldsymbol{L}(\boldsymbol{\theta}) = (\boldsymbol{\Phi}\boldsymbol{\theta} - \mathbf{y})^{\mathrm{T}}(\boldsymbol{\Phi}\boldsymbol{\theta} - \mathbf{y}) + \lambda\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\theta}$$
- By the representer theorem:
$$\boldsymbol{\theta} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$$
- Dual regularized empirical risk:
$$\boldsymbol{L}(\boldsymbol{\alpha}) = \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - \mathbf{y}\right)^{\mathrm{T}}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - \mathbf{y}\right) + \lambda\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$$

Kernel Ridge Regression

- Dual regularized empirical risk:

$$\begin{aligned} \boldsymbol{L}(\boldsymbol{\alpha}) &= \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - \mathbf{y}\right)^{\mathrm{T}}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - \mathbf{y}\right) + \lambda\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} \\ &= \boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - \mathbf{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{y} \\ &\quad + \lambda\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} \end{aligned}$$

- Define gram matrix (or kernel matrix) as $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$.

$$\boldsymbol{L}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{K}\mathbf{K}\boldsymbol{\alpha} - \mathbf{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{K}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{y} + \lambda\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{K}\boldsymbol{\alpha}$$

- Setting the derivative to zero

$$\frac{\partial}{\partial\boldsymbol{\alpha}}\boldsymbol{L}(\boldsymbol{\alpha}) = \mathbf{0}$$

- Gives the solution

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\boldsymbol{I})^{-1}\mathbf{y}$$

Kernel Ridge Regression

- Kernel (gram) matrix: $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}$

$$\mathbf{K} = \begin{pmatrix} - & \phi(\mathbf{x}_1)^{\mathrm{T}} & - \\ \vdots & \ddots & \vdots \\ - & \phi(\mathbf{x}_n)^{\mathrm{T}} & - \end{pmatrix} \begin{pmatrix} | & \dots & | \\ \phi(\mathbf{x}_1) & \ddots & \phi(\mathbf{x}_n) \\ | & \dots & | \end{pmatrix}$$

$$= \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

- $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$

Kernel Ridge Regression

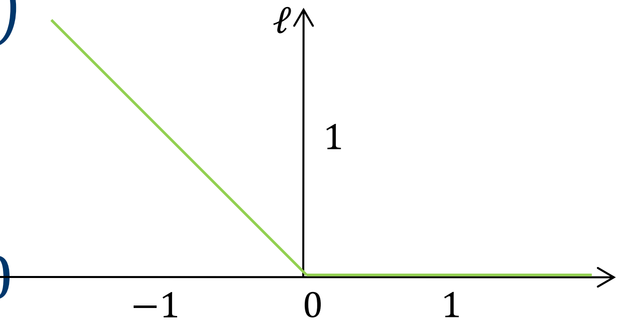
- Regression method that uses kernel functions
- Works with any nonlinear embedding $\boldsymbol{\phi}$ as long as there is a kernel function that computes the inner product: $k(\mathbf{x}_i, \mathbf{x}) = \phi(\mathbf{x}_i)^{\mathrm{T}}\phi(\mathbf{x})$.
- Kernel matrix $\mathbf{K}$ of size $n \times n$ has to be inverted, works only for modest sample sizes.
- Solution dependent on $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, but otherwise independent of $\mathbf{\Phi}$.
- For large sample size, use numeric optimization (e.g., stochastic gradient descent method).

Contents

- Feature mappings
  - Representer Theorem
- Kernel learning algorithms
  - Kernel ridge regression
  - Kernel perceptron,
  - Dual SVM
- Mercer map
- Kernel functions
  - Polynomial, RBF
  - For time series, strings, graphs

Kernel Perceptron

- Loss function:
  $\ell_p(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i) = \max\big(0, -y_i f_{\boldsymbol{\theta}}(\mathbf{x}_i)\big)$
- No regularizer.
- Primal stochastic gradient:
  $$\nabla L_{\mathbf{x}_i}(\boldsymbol{\theta}) = \begin{cases} -y_i \mathbf{x}_i & -y_i f_{\boldsymbol{\theta}}(\mathbf{x}_i) > 0 \\ 0 & -y_i f_{\boldsymbol{\theta}}(\mathbf{x}_i) < 0 \end{cases}$$



Rosenblatt, 1960

Kernel Perceptron

- Stochastic gradient update step:

$$\begin{array}{ll} \texttt{IF} & y_i f_{\boldsymbol{\theta}}(\mathbf{x}_i) \leq 0 \\ \texttt{THEN} & \boldsymbol{\theta}' = \boldsymbol{\theta} + y_i \mathbf{x}_i \end{array}$$

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + y_i \phi(\mathbf{x}_i)$$

$$\Leftrightarrow \sum_{j=1}^{n} \alpha'_j \phi(\mathbf{x}_j) = \sum_{j=1}^{n} \alpha_j \phi(\mathbf{x}_j) + y_i \phi(\mathbf{x}_i)$$

$$\Leftarrow \alpha'_i \phi(\mathbf{x}_i) = \alpha_i \phi(\mathbf{x}_i) + y_i \phi(\mathbf{x}_i),$$

$$\forall j \neq i : \alpha'_j = \alpha_j$$

$$\Leftarrow \alpha'_i = \alpha_i + y_i$$

- Dual stochastic gradient update step:

$$\begin{array}{ll} \texttt{IF} & y_i f_{\boldsymbol{\alpha}}(\mathbf{x}_i) \leq 0 \\ \texttt{THEN} & \alpha_i = \alpha_i + y_i \end{array}$$

Kernel Perceptron Algorithm

```
Perceptron(Instances {(x_i, y_i)})
    Set α = 0
    DO
      FOR i = 1, ..., n
        IF      y_i f_α(x_i) ≤ 0
        THEN    α_i = α_i + y_i
      END
    WHILE α changes
    RETURN α
```

- Decision function:

$$f_{\boldsymbol{\alpha}}(\mathbf{x}) = \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{\Phi} \phi(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$

Kernel Perceptron

- Perceptron loss, no regularizer
- Dual form of the decision function:

$$f_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$

- Dual form of the update rule:
  - If $y_i f_{\boldsymbol{\alpha}}(\mathbf{x}_i) \leq 0$, then $\alpha_i = \alpha_i + y_i$
- Equivalent to the primal form of the perceptron
- Advantageous to use instead of the primal perceptron if there are few samples and $\phi(\mathbf{x})$ is high dimensional.

Contents

- Feature mappings
  - Representer Theorem
- Kernel learning algorithms
  - Kernel ridge regression
  - Kernel perceptron,
  - Dual SVM
- Mercer map
- Kernel functions
  - Polynomial, RBF
  - For time series, strings, graphs

Kernel Support Vector Machine

- Primal: $\min_{\boldsymbol{\theta}} \left[\sum_{i=1}^{n} \max\left(0, 1 - y_i \phi(\mathbf{x}_i)^{\mathrm{T}} \boldsymbol{\theta}\right) + \frac{1}{2\lambda} \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}\right]$

- Equivalent optimization problem with side constraints:

$$\min_{\boldsymbol{\theta}, \boldsymbol{\xi}} \left[\lambda \sum_{i=1}^{n} \xi_i + \frac{1}{2} \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}\right]$$

such that

$$y_i \phi(\mathbf{x}_i)^{\mathrm{T}} \boldsymbol{\theta} \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

- Goal: dual formulization of the optimization problem

Kernel Support Vector Machine

- Optimization problem with side constraints:

$$\min_{\boldsymbol{\theta},\boldsymbol{\xi}} \left[\lambda \sum_{i=1}^{n} \xi_i + \frac{1}{2}\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\theta}\right]$$

such that

$$y_i \phi(\mathbf{x}_i)^{\mathrm{T}}\boldsymbol{\theta} \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

Goal function: $Z(\boldsymbol{\theta}, \boldsymbol{\xi})$
Side constraints: $g(\boldsymbol{\theta}, \boldsymbol{\xi}) \geq 0$
Lagrange function: $Z(\boldsymbol{\theta}, \boldsymbol{\xi}) - \beta g(\boldsymbol{\theta}, \boldsymbol{\xi})$

- Lagrange function with Lagrange-Multipliers $\boldsymbol{\beta} \geq \mathbf{0}$ and $\boldsymbol{\beta}^0 \geq \mathbf{0}$ for the side constraints:

$$L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\beta}^0) = \lambda \sum_{i=1}^{n} \xi_i + \frac{\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\theta}}{2} - \sum_{i=1}^{n} \beta_i \left(y_i \phi(\mathbf{x}_i)^{\mathrm{T}}\boldsymbol{\theta} - 1 + \xi_i\right) - \sum_{i=1}^{n} \beta_i^0 \xi_i$$

- Optimization problem without side constraints:

$$\min_{\boldsymbol{\theta},\boldsymbol{\xi}} \max_{\boldsymbol{\beta},\boldsymbol{\beta}^0} L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\beta}^0)$$

Kernel Support Vector Machine

- Lagrange function:

$$L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\beta}^0) = \lambda \sum_{i=1}^{n} \xi_i + \frac{\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\theta}}{2} - \sum_{i=1}^{n} \beta_i \big(y_i \phi(\mathbf{x}_i)^{\mathrm{T}}\boldsymbol{\theta} - 1 + \xi_i\big) - \sum_{i=1}^{n} \beta_i^0 \xi_i$$

- Setting the derivative of L w.r.t. $(\boldsymbol{\theta}, \boldsymbol{\xi})$ to zero gives:

$$\frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\beta}^0) = \mathbf{0} \;\Rightarrow\; \boldsymbol{\theta} = \textstyle\sum_{i=1}^{n} \underbrace{\beta_i y_i}_{\alpha_i} \phi(\mathbf{x}_i)$$

$$\frac{\partial}{\partial \xi_i} L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\beta}^0) = 0 \;\Rightarrow\; \lambda = \beta_i + \beta_i^0$$

Relation between primal and dual parameters… representer theorem.

Kernel Support Vector Machine

$$\boldsymbol{\theta} = \sum_{i=1}^{n} \beta_i y_i \phi(\mathbf{x}_i)$$
$$\lambda = \beta_i + \beta_i^0$$

- Substitute the derived parameters into the Lagrange function:

$$L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\beta}^0) = \frac{1}{2}(\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{\theta})$$
$$-\sum_{i=1}^{n} \beta_i \left(y_i \phi(\mathbf{x}_i)^{\mathrm{T}} \boldsymbol{\theta} - 1 + \xi_i\right) - \sum_{i=1}^{n} \beta_i^0 \xi_i + \lambda \sum_{i=1}^{n} \xi_i$$

Kernel Support Vector Machine

$$\boldsymbol{\theta} = \sum_{i=1}^{n} \beta_i y_i \phi(\mathbf{x}_i)$$
$$\lambda = \beta_i + \beta_i^0$$

- Substitute the derived parameters into the Lagrange function:

$$L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\beta}^0) = \frac{1}{2}\left(\sum_{i=1}^{n} \beta_i y_i \phi(\mathbf{x}_i)\right)^{\mathrm{T}} \left(\sum_{j=1}^{n} \beta_j y_j \phi(\mathbf{x}_j)\right)$$
$$- \sum_{i=1}^{n} \beta_i \left(y_i \phi(\mathbf{x}_i)^{\mathrm{T}} \sum_{j=1}^{n} \beta_j y_j \phi(\mathbf{x}_j) - 1 + \xi_i \right) - \sum_{i=1}^{n} \beta_i^0 \xi_i + \lambda \sum_{i=1}^{n} \xi_i$$

Kernel Support Vector Machine

$$\boldsymbol{\theta} = \sum_{i=1}^{n} \beta_i y_i \phi(\mathbf{x}_i)$$
$$\lambda = \beta_i + \beta_i^0$$

- Substitute the derived parameters into the Lagrange function:

$$L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\beta}^0) = \frac{1}{2}\left(\sum_{i=1}^{n} \beta_i y_i \phi(\mathbf{x}_i)\right)^{\mathrm{T}} \left(\sum_{j=1}^{n} \beta_j y_j \phi(\mathbf{x}_j)\right)$$
$$- \sum_{i=1}^{n} \beta_i \left(y_i \phi(\mathbf{x}_i)^{\mathrm{T}} \sum_{j=1}^{n} \beta_j y_j \phi(\mathbf{x}_j) - 1 + \xi_i \right) - \sum_{i=1}^{n} \beta_i^0 \xi_i + \lambda \sum_{i=1}^{n} \xi_i$$
$$= \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x}_j)$$
$$- \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x}_j) + \sum_{i=1}^{n} \beta_i - \sum_{i=1}^{n} \underbrace{\left(\beta_i + \beta_i^0\right)}_{=\lambda} \xi_i + \lambda \sum_{i=1}^{n} \xi_i$$

Kernel Support Vector Machine

$$\boldsymbol{\theta} = \sum\nolimits_{i=1}^{n} \beta_i y_i \phi(\mathbf{x}_i)$$
$$\lambda = \beta_i + \beta_i^0$$

- Substitute the derived parameters into the Lagrange function:

$$L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\beta}, \boldsymbol{\beta}^0) = \frac{1}{2}\left(\sum_{i=1}^{n} \beta_i y_i \phi(\mathbf{x}_i)\right)^{\mathrm{T}} \left(\sum_{j=1}^{n} \beta_j y_j \phi(\mathbf{x}_j)\right)$$
$$- \sum_{i=1}^{n} \beta_i \left(y_i \phi(\mathbf{x}_i)^{\mathrm{T}} \sum_{j=1}^{n} \beta_j y_j \phi(\mathbf{x}_j) - 1 + \xi_i \right) - \sum_{i=1}^{n} \beta_i^0 \xi_i + \lambda \sum_{i=1}^{n} \xi_i$$
$$= \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x}_j)$$
$$- \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x}_j) + \sum_{i=1}^{n} \beta_i - \sum_{i=1}^{n} \underbrace{(\beta_i + \beta_i^0)}_{=\lambda} \xi_i + \lambda \sum_{i=1}^{n} \xi_i$$
$$= \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x}_j)$$

Kernel Support Vector Machine

- Optimization criterion of the dual SVM:

$$\max_{\boldsymbol{\beta}} \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

such that

$$0 \leq \beta_i \leq \lambda$$

L1-Regularizer of $\boldsymbol{\beta}$ (sparse)

Large if $\beta_i, \beta_j > 0$ for similar instances of different classes.

Kernel Support Vector Machine

- Optimization criterion of the dual SVM:

$$\max_{\boldsymbol{\beta}} \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right)$$

- Optimization over parameters $\boldsymbol{\beta}$.
- Solution found with QP-Solver in $O(n^2)$.
- Sparse solution.
- Samples only appear as pairwise inner products.

Kernel Support Vector Machine

- Primal and dual optimization problem have the same solution.

$$\boldsymbol{\theta} = \sum_{\mathbf{x}_i \in SV} \beta_i y_i \phi(\mathbf{x}_i)$$

Support Vectors:
$\beta_i > 0$

- Dual form of the decision function:

$$f_{\boldsymbol{\beta}}(\mathbf{x}) = \sum_{\mathbf{x}_i \in SV} \beta_i y_i k(\mathbf{x}_i, \mathbf{x})$$

- Primal SVM:
  - Solution is a Vector $\boldsymbol{\theta}$ in the space of the attributes.
- Dual SVM:
  - The same solution is represented as weights β_i of the samples.

Constructing Kernels

- Design embedding $\phi(\mathbf{x})$, then obtain resulting kernel function $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{x}')$.
- Or: just define kernel function (any similarity measure) $k(\mathbf{x}, \mathbf{x}')$ directly, don't bother with embedding.
- For which functions k does there exist a mapping $\phi(\mathbf{x})$, so that k represents an inner product?

Kernels

- Kernel matrices are symmetric:
$$\mathbf{K} = \mathbf{K}^{\mathrm{T}}$$
- Kernel matrices $\mathbf{K} \in \mathbb{R}^{n \times n}$ are positive semidefinite:
$$\exists \mathbf{\Phi} \in \mathbb{R}^{n \times m}: \mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}$$
- Kernel function $k(\mathbf{x}, \mathbf{x}')$ is positive semidefinite if $\mathbf{K}$ is positive semidefinite for every data set.

- For every positive definite function k there is at least one mapping $\phi(\mathbf{x})$ such that $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{x}')$ for all $\mathbf{x}$ and $\mathbf{x}'$.

Contents

- Feature mappings
  - Representer Theorem
- Kernel learning algorithms
  - Kernel ridge regression
  - Kernel perceptron,
  - Dual SVM
- Mercer map
- Kernel functions
  - Polynomial, RBF
  - For time series, strings, graphs

Mercer Map

- Eigenvalue decomposition: Every symmetric matrix $\mathbf{K}$ can be decomposed in terms of its eigenvectors $\mathbf{u}_i$ and eigenvalues λ_i:

$$\mathbf{K} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}, \text{ with } \mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{pmatrix} \,\&\, \mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_n \\ | & & | \end{pmatrix}$$

- If $\mathbf{K}$ is positive semi-definite, then $\lambda_i \in \mathbb{R}^{0+}$
- The eigenvectors are orthonormal ($\mathbf{u}_i^{\mathrm{T}}\mathbf{u}_i = 1$ and $\mathbf{u}_i^{\mathrm{T}}\mathbf{u}_j = 0$) and $\mathbf{U}$ is orthogonal: $\mathbf{U}^{\mathrm{T}} = \mathbf{U}^{-1}$.

Mercer Map

Eigenvalue decomposition

- Thus it holds:

Diagonal matrix with $\sqrt{\lambda_i}$

$$
\begin{aligned}
\mathbf{K} &= \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{T}} \\
&= \left(\mathbf{U}\boldsymbol{\Lambda}^{1/2}\right)\left(\boldsymbol{\Lambda}^{1/2}\mathbf{U}^{\mathrm{T}}\right) \\
&= \left(\mathbf{U}\boldsymbol{\Lambda}^{1/2}\right)\left(\mathbf{U}\boldsymbol{\Lambda}^{1/2}\right)^{\mathrm{T}}
\end{aligned}
$$

- Feature mapping for training data can be defined as

$$
\begin{pmatrix} | & & | \\ \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_n) \\ | & & | \end{pmatrix} = \left(\mathbf{U}\boldsymbol{\Lambda}^{1/2}\right)^{\mathrm{T}}
$$

Mercer Map

- Feature mapping for used training data can then be defined as

$$\begin{pmatrix} | & & | \\ \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_n) \\ | & & | \end{pmatrix} = \left(\mathbf{U}\boldsymbol{\Lambda}^{1/2}\right)^{\mathrm{T}}$$

- Kernel matrix between training and test data

$$\begin{aligned} \mathbf{K}_{test} &= \Phi(\mathbf{X}_{train})^{\mathrm{T}}\Phi(\mathbf{X}_{test}) \\ &= \left(\mathbf{U}\boldsymbol{\Lambda}^{1/2}\right)\Phi(\mathbf{X}_{test}) \end{aligned}$$

- Equation results in a mapping of the test data:

$$\Phi(\mathbf{X}_{test}) = \left(\mathbf{U}\boldsymbol{\Lambda}^{1/2}\right)^{-1}\mathbf{K}_{test}$$

$$\Phi(\mathbf{X}_{test}) = \boldsymbol{\Lambda}^{-1/2}\mathbf{U}^{\mathrm{T}}\mathbf{K}_{test}$$

$\boxed{\mathbf{U}^{\mathrm{T}} = \mathbf{U}^{-1}}$

Mercer Map

- Useful if a learning problem is given as a kernel function but learning should take place in the primal.
- For example if the kernel matrix will be too large (quadratic memory consumption!)

Contents

- Feature mappings
  - Representer Theorem
- Kernel learning algorithms
  - Kernel ridge regression
  - Kernel perceptron,
  - Dual SVM
- Mercer map
- Kernel functions
  - Polynomial, RBF
  - For time series, strings, graphs

Kernel Compositions

- Kernel functions can be composed:

$$k(\mathbf{x}, \mathbf{x}') = c k_1(\mathbf{x}, \mathbf{x}')$$
$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')$$
$$k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}'))$$
$$k(\mathbf{x}, \mathbf{x}') = e^{k_1(\mathbf{x}, \mathbf{x}')}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$$

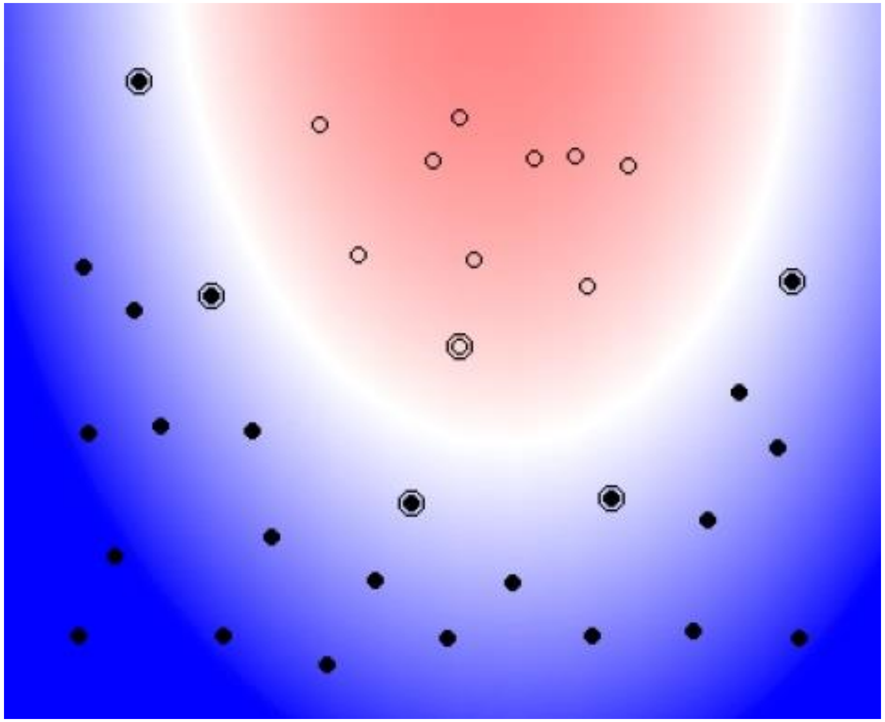
...

Kernel Functions

- Polynomial kernels: $k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j + 1)^p$
- Radial basis functions: $k_{RBF}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$
- Sigmoid kernels,
- Dynamic time-warping kernels,
- String kernels,
- Graph kernels,

Polynomial Kernels

- Kernel function: $k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1\right)^p$
- Which transformation ϕ corresponds to this kernel?
- Example: 2-D input space, $p = 2$.



Polynomial Kernels

- Kernel: $k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1\right)^p$, 2D-input, $p = 2$.

$$k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1\right)^2$$

$$= \left(\begin{pmatrix}\mathbf{x}_{i1} & \mathbf{x}_{i2}\end{pmatrix}\begin{pmatrix}\mathbf{x}_{j1} \\ \mathbf{x}_{j2}\end{pmatrix} + 1\right)^2 = \left(\mathbf{x}_{i1}\mathbf{x}_{j1} + \mathbf{x}_{i2}\mathbf{x}_{j2} + 1\right)^2$$

Polynomial Kernels

- Kernel: $k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1)^p$, 2D-input, $p = 2$.

$$k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1)^2$$

$$= \left(\begin{pmatrix} \mathbf{x}_{i1} & \mathbf{x}_{i2} \end{pmatrix} \begin{pmatrix} \mathbf{x}_{j1} \\ \mathbf{x}_{j2} \end{pmatrix} + 1 \right)^2 = (\mathbf{x}_{i1}\mathbf{x}_{j1} + \mathbf{x}_{i2}\mathbf{x}_{j2} + 1)^2$$

$$= (\mathbf{x}_{i1}^2\mathbf{x}_{j1}^2 + \mathbf{x}_{i2}^2\mathbf{x}_{j2}^2 + 2\mathbf{x}_{i1}\mathbf{x}_{j1}\mathbf{x}_{i2}\mathbf{x}_{j2} + 2\mathbf{x}_{i1}\mathbf{x}_{j1} + \mathbf{2}\mathbf{x}_{i2}\mathbf{x}_{j2} + 1)$$

$$= \underbrace{\begin{pmatrix} \mathbf{x}_{i1}^2 & \mathbf{x}_{i2}^2 & \sqrt{2}\mathbf{x}_{i1}\mathbf{x}_{i2} & \sqrt{2}\mathbf{x}_{i1} & \sqrt{2}\mathbf{x}_{i2} & 1 \end{pmatrix}}_{\substack{\phi(\mathbf{x}_i)^{\mathrm{T}} \\ \text{All monomials of degree} \leq 2 \text{ over input attributes}}} \underbrace{\begin{pmatrix} \mathbf{x}_{j1}^2 \\ \mathbf{x}_{j2}^2 \\ \sqrt{2}\mathbf{x}_{j1}\mathbf{x}_{j2} \\ \sqrt{2}\mathbf{x}_{j1} \\ \sqrt{2}\mathbf{x}_{j2} \\ 1 \end{pmatrix}}_{\phi(\mathbf{x}_j)}$$

Polynomial Kernels

- Kernel: $k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1\right)^p$, 2D-input, $p = 2$.

$$k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1\right)^2$$

$$= \left(\begin{pmatrix}\mathbf{x}_{i1} & \mathbf{x}_{i2}\end{pmatrix}\begin{pmatrix}\mathbf{x}_{j1} \\ \mathbf{x}_{j2}\end{pmatrix} + 1\right)^2 = \left(\mathbf{x}_{i1}\mathbf{x}_{j1} + \mathbf{x}_{i2}\mathbf{x}_{j2} + 1\right)^2$$

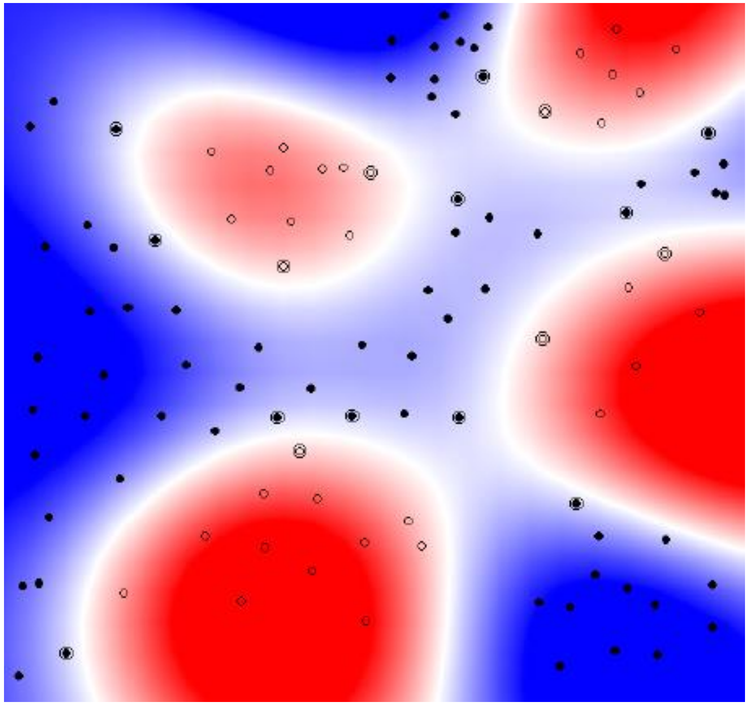
$$= \left(\mathbf{x}_{i1}^2\mathbf{x}_{j1}^2 + \mathbf{x}_{i2}^2\mathbf{x}_{j2}^2 + 2\mathbf{x}_{i1}\mathbf{x}_{j1}\mathbf{x}_{i2}\mathbf{x}_{j2} + 2\mathbf{x}_{i1}\mathbf{x}_{j1} + \mathbf{2}\mathbf{x}_{i2}\mathbf{x}_{j2} + 1\right)$$

$$= \underbrace{\begin{pmatrix}\mathbf{x}_{i1}^2 & \mathbf{x}_{i2}^2 & \sqrt{2}\mathbf{x}_{i1}\mathbf{x}_{i2} & \sqrt{2}\mathbf{x}_{i1} & \sqrt{2}\mathbf{x}_{i2} & 1\end{pmatrix}}_{\substack{\phi(\mathbf{x}_i)^{\mathrm{T}} \\ \text{All monomials of degree} \leq 2 \text{ over input attributes}}} \underbrace{\begin{pmatrix}\mathbf{x}_{j1}^2 \\ \mathbf{x}_{j2}^2 \\ \sqrt{2}\mathbf{x}_{j1}\mathbf{x}_{j2} \\ \sqrt{2}\mathbf{x}_{j1} \\ \sqrt{2}\mathbf{x}_{j2} \\ 1\end{pmatrix}}_{\phi(\mathbf{x}_j)}$$

$$= \begin{pmatrix}\mathbf{x}_i \otimes \mathbf{x}_i \\ \sqrt{2}\mathbf{x}_i \\ 1\end{pmatrix}^T \begin{pmatrix}\mathbf{x}_j \otimes \mathbf{x}_j \\ \sqrt{2}\mathbf{x}_j \\ 1\end{pmatrix}$$

RBF Kernel

- Kernel: $k_{RBF}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right)$
- No finite-dimensional feature mapping ϕ.



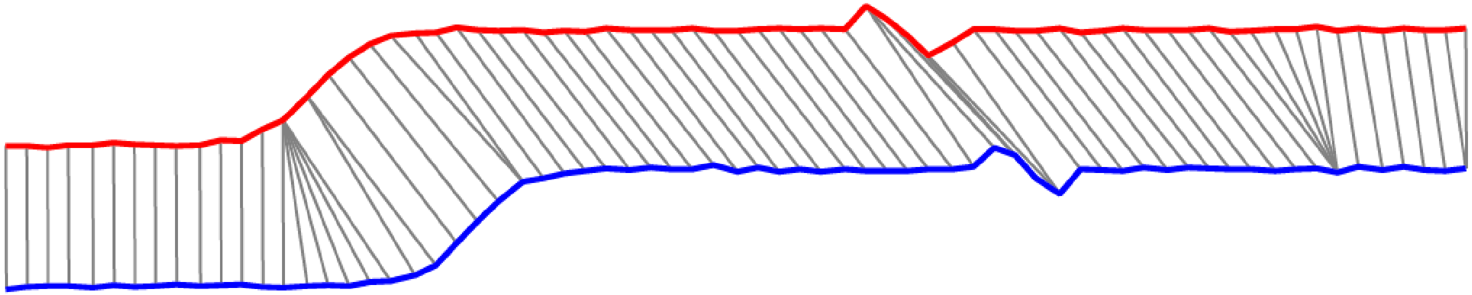
Time Series: DTW Kernel

- Similarity of time series
- Idea: Find corresponding similar points in $\mathbf{x}, \mathbf{x}'$.
- Correspondence function

$$\pi_{\mathbf{x}}(k) \in [1, T_{\mathbf{x}}], \pi_{\mathbf{x}'}(l) \in [1, T_{\mathbf{x}'}]$$

- DTW distance is squared distance between matched sequences:

$$k_{DTW}(\mathbf{x}, \mathbf{x}') = e^{-\left(\min \sum_{k=1}^{T} \left(\mathbf{x}_{\pi_{\mathbf{x}(k)}} - \mathbf{x}'_{\pi_{\mathbf{x}'(k)}}\right)^2\right)}$$



Time Series: DTW Kernel

- Efficient calculation using dynamic programming
- Let $\gamma(k,l)$ be the minimum squared distance of corresponding points up to time k and l.
- Recursive update:
$$\gamma(k,l) = (\mathbf{x}_k - \mathbf{x}_l)^2 + \min\{\gamma(k-1,l-1), \gamma(k-1,l), \gamma(k,l-1)\}$$
- Algorithm:

DTW(*Sequences* $\mathbf{x}$ *and* $\mathbf{x}$‘)

Let $\gamma(0,0) = 0; \gamma(k,0) = \infty; \gamma(0,l) = \infty$

FOR $k = 1 \ldots T_x$

FOR $l = 1 \ldots T_y$

$\gamma(k,l) = (\mathbf{x}_k - \mathbf{x}_l)^2 + \min\{\gamma(k-1,l-1), \gamma(k-1,l), \gamma(k,l-1)\}$

RETURN $\gamma(T_x, T_y)$

Intelligent Data Analysis

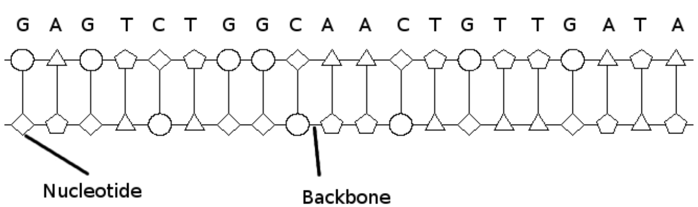
Strings: Motivation

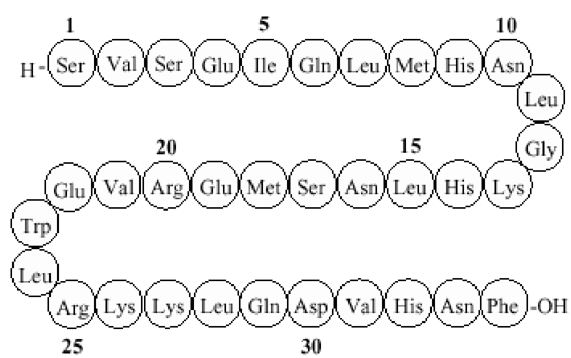
- Strings are a common non-numeric type of data
  - Documents & email are strings

From: Webmaster Admin <in-foweb@live.co.uk>
To: undisclosed-recipients: ;
Reply-to: in-foweb@live.co.uk
Subject: Attention !! Re-activer le service e-mail
Date: Wed, 19 Jan 2011 15:54:21 +0100 (CET)
User-Agent: SquirrelMail/1.4.8-5.el5.centos.10

Votre quota a dépassé l'ensemble quota/limite est de 20 Go Vous êtes en cours d'exécution sur 23FR de fichiers et parce que les fichiers cachés sur votre e-mail.

  - DNA & Protein sequences are strings





String Kernels

- String – a sequence of characters from alphabet Σ written as $\mathbf{s} = s_1 s_2 \dots s_n$ with $|\mathbf{s}| = n$.
  - The set of all strings is $\Sigma^* = \bigcup_{n \in N} \Sigma^n$
  - $\mathbf{s}_{i:j} = s_i s_{i+1} \dots s_j$
  - Subsequence: for any $\mathbf{i} \in \{0,1\}^n$, $\mathbf{s}[\mathbf{i}]$ is the elements of $\mathbf{s}$ corresponding to elements of $\mathbf{i}$ that are 1
    - Eg. If $\mathbf{s}$="abcd" $\mathbf{s}[(1,0,0,1)]$="ad"

- A string kernel is a real-valued function on $\Sigma^* \times \Sigma^*$.
  - We need positive definite kernels
  - We will design kernels by looking at a feature space of substrings / subsequences

Bag-of-Words Kernel

- For textual data, a simple feature representation is indexed by the words contained in the string

Email

> Dear Beneficiary,
>
> your Email address has been picked online in this years MICROSOFT CONSUMER AWARD as a Winner of One Hundred and Fifty Five Thousand Pounds Sterling…

Attribute

Word #1 occurs?

⋮

Word #m occurs?

$m \approx 1{,}000{,}000$

Instance $\mathbf{x}$

$$\begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 1 \\ 0 \end{pmatrix} \begin{array}{l} \text{Aardvark} \\ \text{Beneficiary} \\ \text{Friend} \\ \vdots \\ \text{Sterling} \\ \text{Science} \end{array}$$

- Bag-of-Words Kernel computes the number of common words between 2 texts; efficient?

Spectrum Kernel

- Consider feature space with features corresponding to every p length substring of alphabet Σ.
  - $\phi(\mathbf{s})_{\mathbf{u}}$ is # of times $\mathbf{u} \in \Sigma^p$ is contained in string $\mathbf{s}$
- The p-spectrum kernel is the result

$$\kappa_p(\mathbf{s}, \mathbf{t}) = \sum_{\mathbf{u} \in \Sigma^p} \phi(\mathbf{s})_{\mathbf{u}}{}^{\mathrm{T}} \phi(\mathbf{t})_{\mathbf{u}}$$

ϕ	aa	ab	ba	bb
aaab	2	1	0	0
bbab	0	1	1	1
aaaa	3	0	0	0
baab	1	1	1	0

K	aaab	bbab	aaaa	baab
aaab	5	1	6	3
bbab	1	3	0	2
aaaa	6	0	9	3
baab	3	2	3	3

Spectrum Kernel – Computation

- Without explicitly computing this feature map, the p-spectrum kernel can be computed as

$$\kappa_p(\mathbf{s}, \mathbf{t}) = \sum_{i=1}^{|\mathbf{s}|-p+1} \sum_{j=1}^{|\mathbf{t}|-p+1} \mathrm{I}[\![\mathbf{s}_{i:i+p-1} = \mathbf{t}_{j:j+p-1}]\!]$$

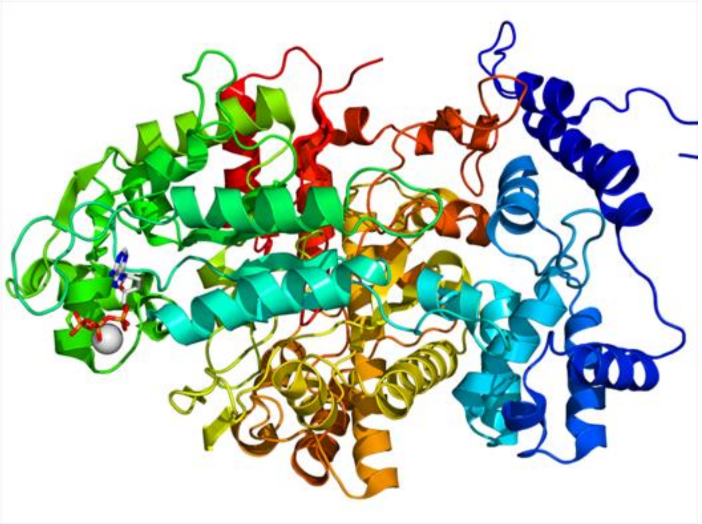
  - This computation is $O(p|\mathbf{s}||\mathbf{t}|)$.
  - Using trie data structures, this computation can be reduced to $O\big(p \cdot max(|\mathbf{s}|, |\mathbf{t}|)\big)$.
- Naturally, we can also compute (weighted) sums of different length substrings

String Kernels

- All-subsequences kernel determines the number of subsequences that appear in both strings
- Fixed-length subsequence kernels
- Gap-weighted subsequence kernels…

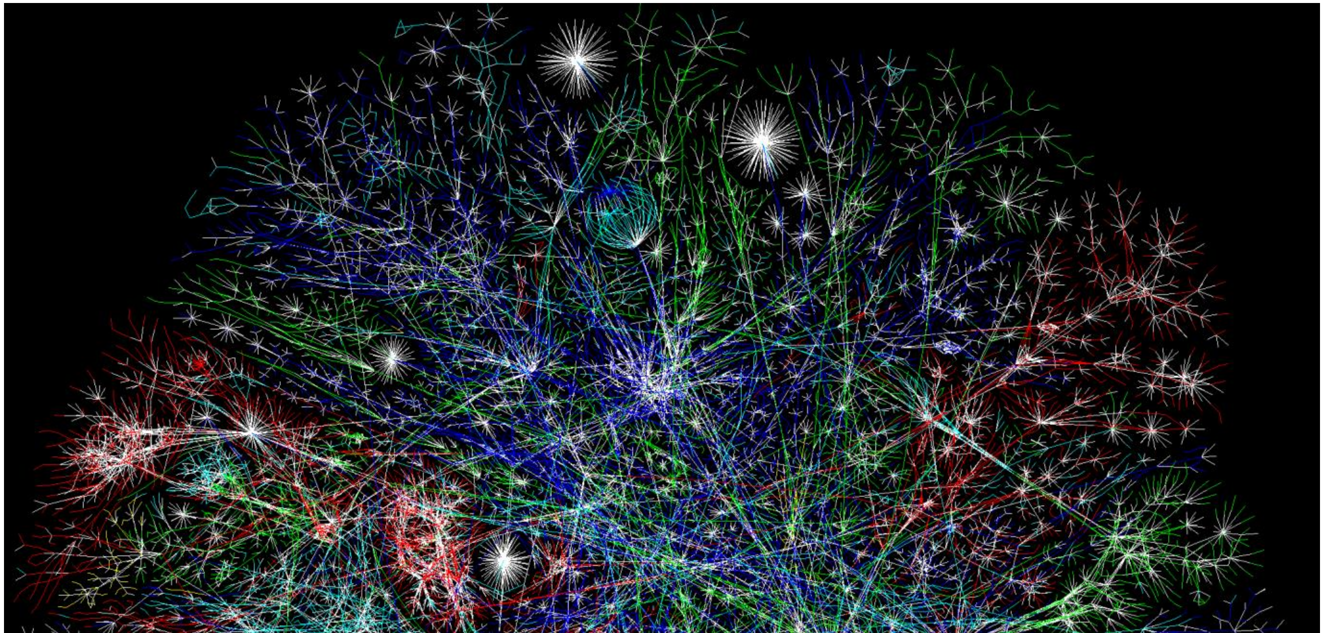
Graphs: Motivation

- Graphs are often used to model objects and their relationship to one another:
  - Bioinformatics: Molecule relationships
  - Internet, social networks
  - …
- Central Question:
  - How similar are two Graphs?
  - How similar are two nodes within a Graph?



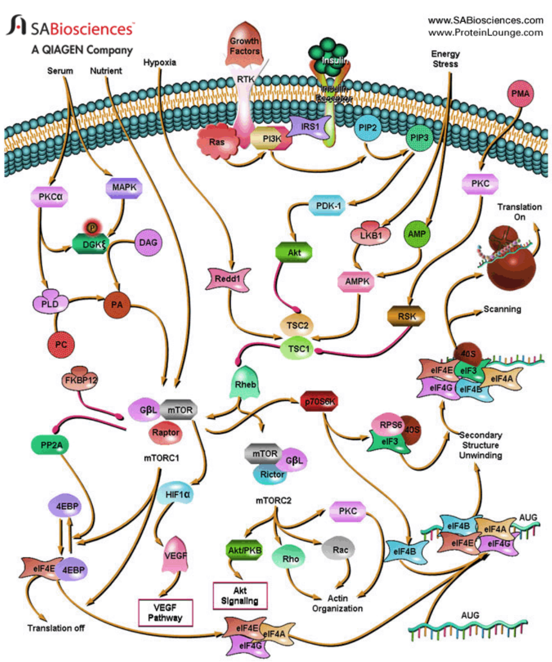
Graph Kernel: Example

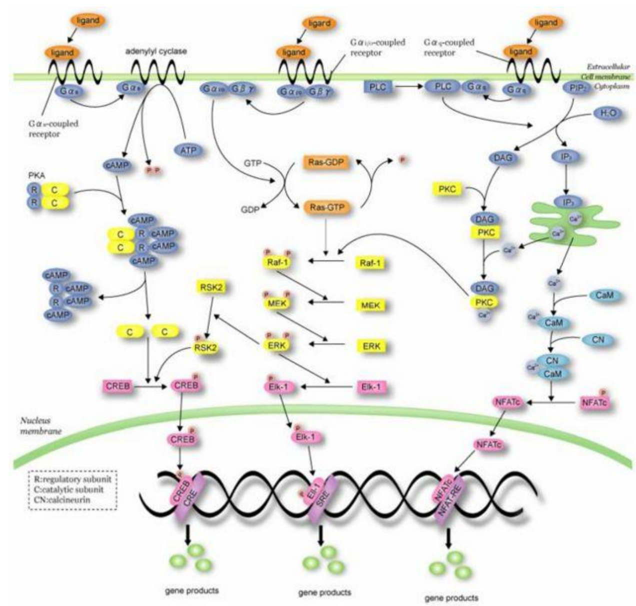
- Consider a dataset of websites with links constituting the edges in the graph
  - A kernel on the nodes of the graph would be useful for learning w.r.t. the web-pages
  - A kernel on graphs would be useful for comparing different components of the internet (e.g. domains)



Graph Kernel: Example

- Consider a set of chemical pathways (sequences of interactions among molecules); i.e. graphs
  - A node kernel would a useful way to measure similarity of different molecules’ roles within these
  - A graph kernel would be a useful measure of similarity for different pathways

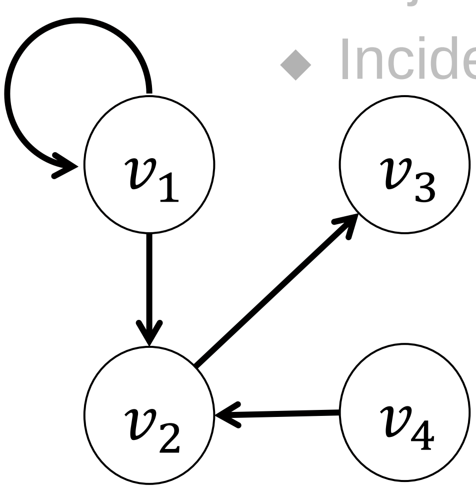




Graphs: Definition

- A graph $G = (V, E)$ is specified by
  - A set of nodes: $v_1, \ldots, v_n \in V$
  - A set of edges: $E \subseteq V \times V$

- Data structures for representing graphs:
  - Adjacency matrix: $\mathbf{A} = \left(a_{ij}\right)_{i,j=1}^{n}, \;\; a_{ij} = \mathrm{I}[\![(v_i, v_j) \in E]\!]$
  - Adjacency list
  - Incidence matrix

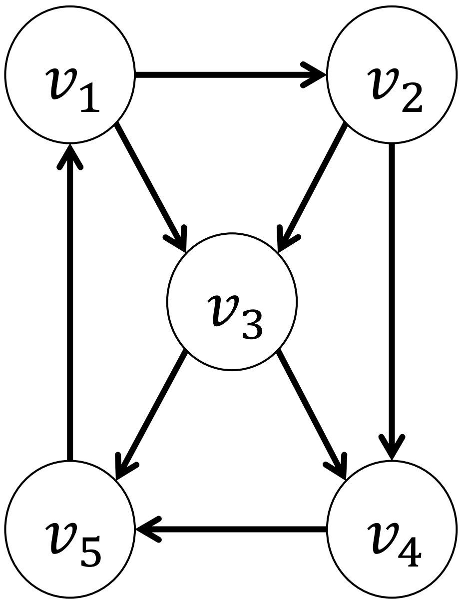


$$G_1 = (V_1, E_1)$$
$$V_1 = \{v_1, \ldots, v_4\}$$
$$E_1 = \begin{Bmatrix} (v_1, v_1), (v_1, v_2), \\ (v_2, v_3), (v_4, v_2) \end{Bmatrix}$$

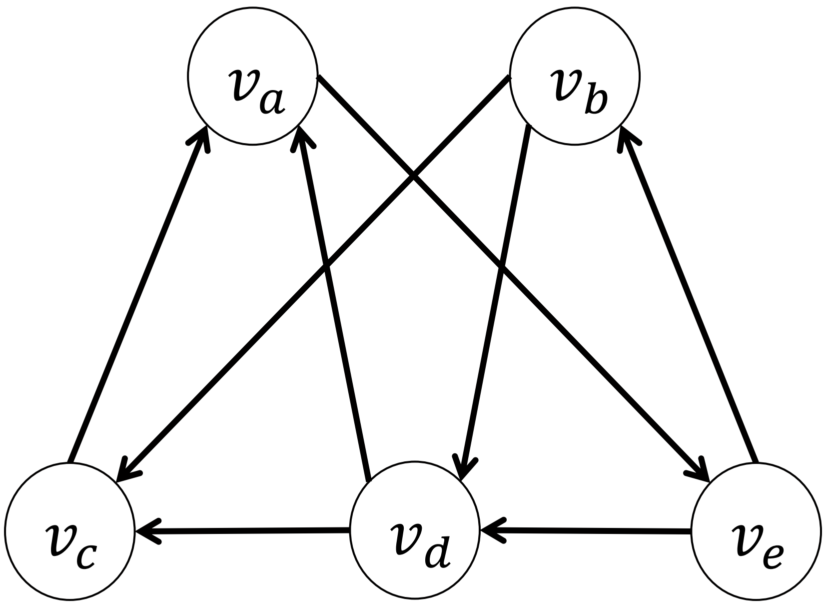
$$\mathbf{A}_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Similarity between Graphs

- Central Question: How similar are two graphs?
- 1st Possibility: Number of isomorphisms between all (sub-) graphs.



$G_1 = (V_1, E_1)$

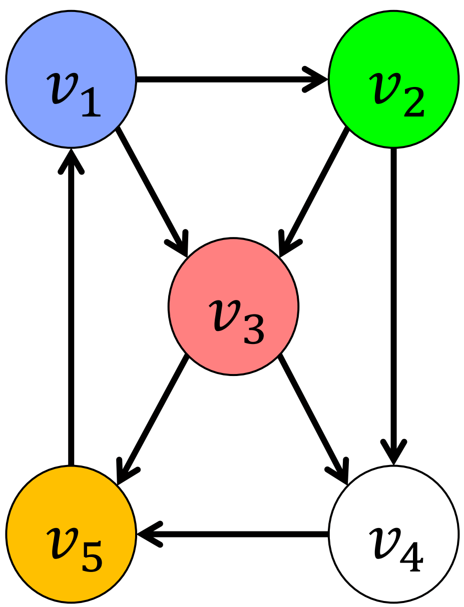


$G_2 = (V_2, E_2)$

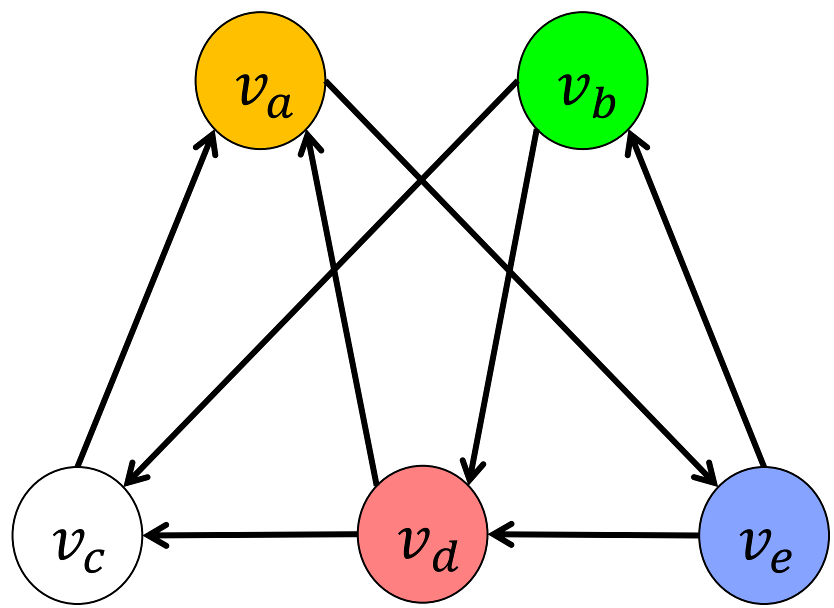
Isomorphisms of Graphs

- Isomorphism: Two Graphs $G_1 = (V_1, E_1)$ & $G_2 = (V_2, E_2)$ are isomorphic if there exists a bijective mapping $f : V_1 \rightarrow V_2$ so that

$$(v_i, v_j) \in E_1 \;\Rightarrow\; \left(f(v_i), f(v_j)\right) \in E_2$$



$G_1 = (V_1, E_1)$



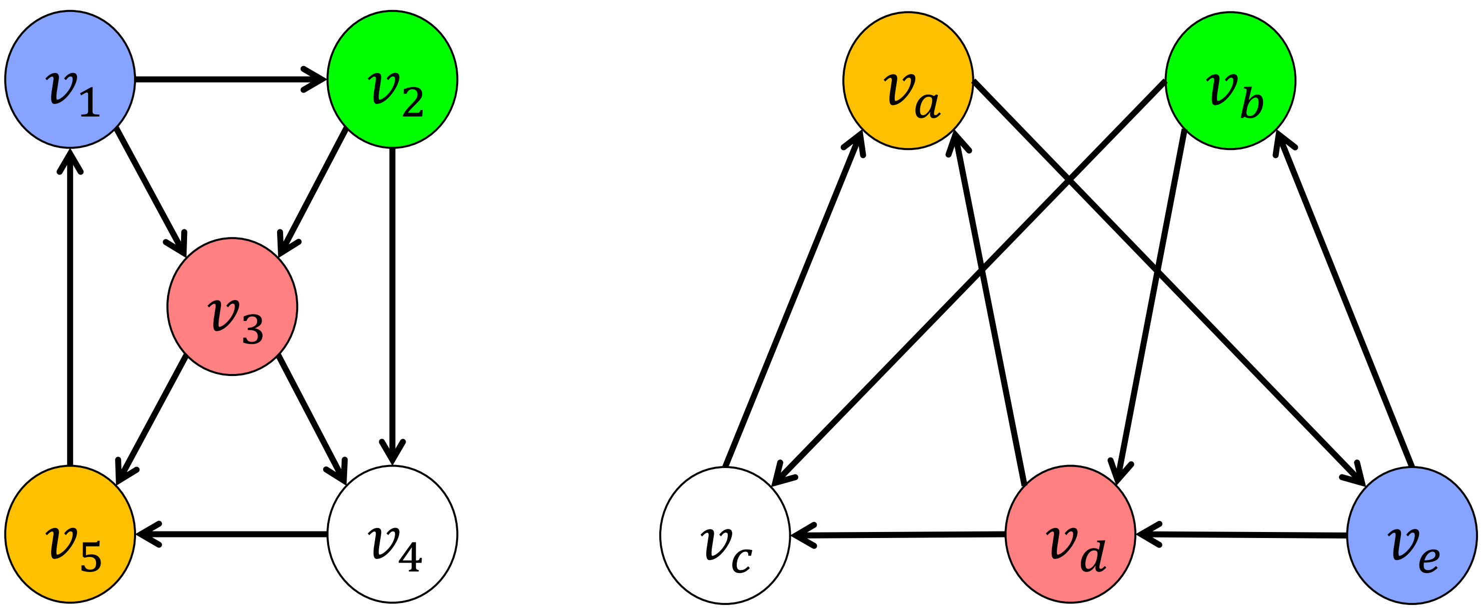
$G_2 = (V_2, E_2)$

Isomorphisms of Graphs

- Isomorphism: Two Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a bijective mapping $f : V_1 \rightarrow V_2$ with

$$\left(v_i, v_j\right) \in E_1 \;\Rightarrow\; \left(f(v_i), f(v_j)\right) \in E_2$$

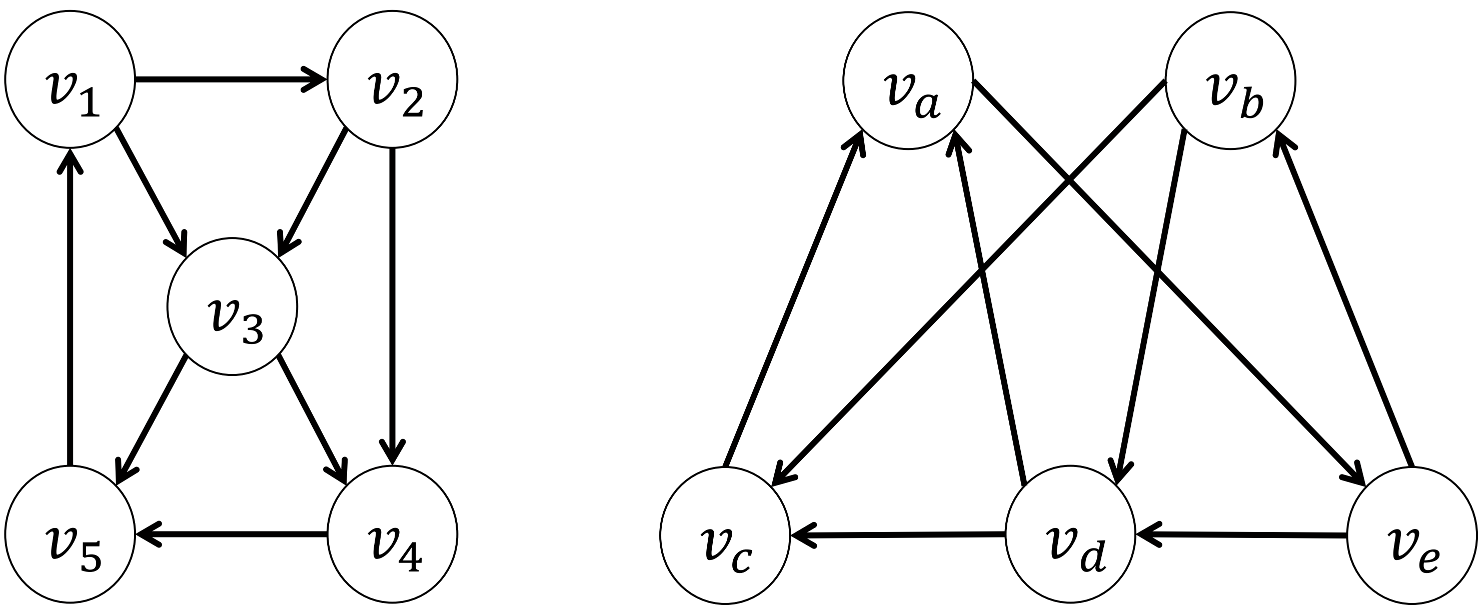
Subgraph isomorphism:
NP-hard!



$G_1 = (V_1, E_1)$ $\qquad$ $G_2 = (V_2, E_2)$

Similarity between Graphs

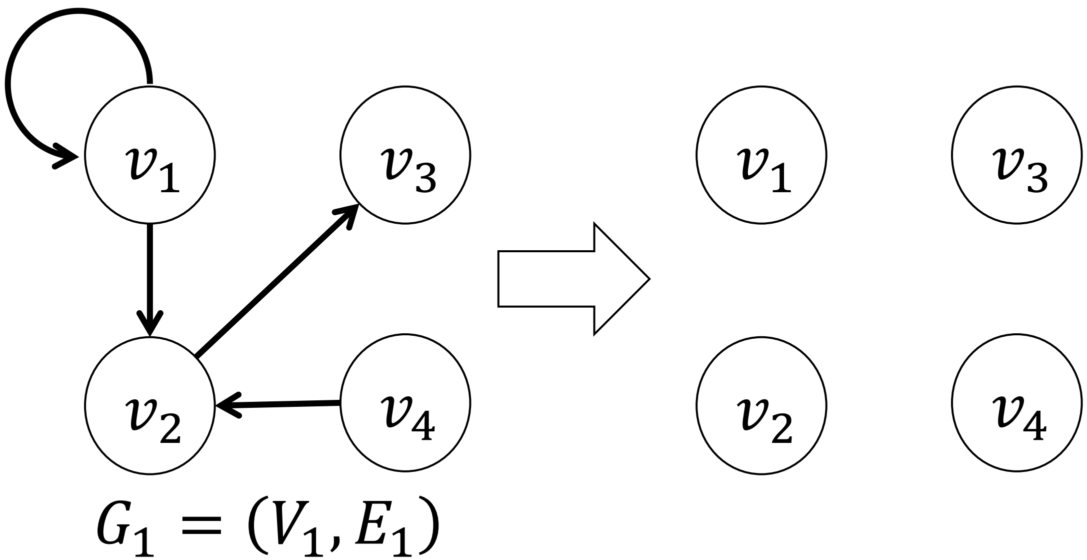
- Central Question: How similar are two graphs?
- 2nd Possibility: Counting the number of “common” paths in the graph.



$G_1 = (V_1, E_1)$ $G_2 = (V_2, E_2)$

Common Paths in Graphs

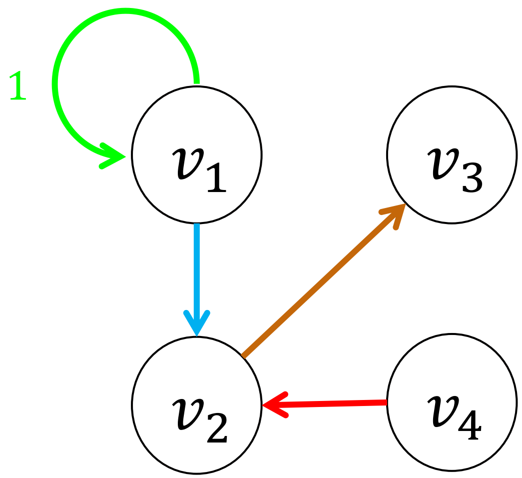
- The number of paths of length 0 is just the number of nodes in the graph.



Intelligent Data Analysis

Common Paths in Graphs

- The number of paths of length 1 from one node to any other is given by the adjacency matrix.



$G_1 = (V_1, E_1)$

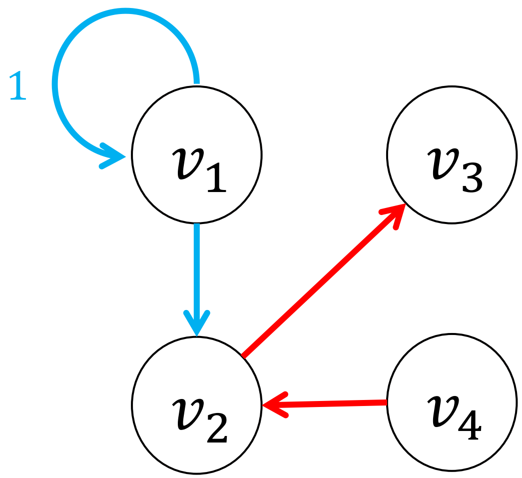
From

$$\mathbf{A}_1 = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \underbrace{\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}}_{v_1 \quad v_2 \quad v_3 \quad v_4}$$

To

Common Paths in Graphs

- Number of paths of length k from one node to any other are given by the k^{th} power of the adjacency matrix.



$G_1 = (V_1, E_1)$

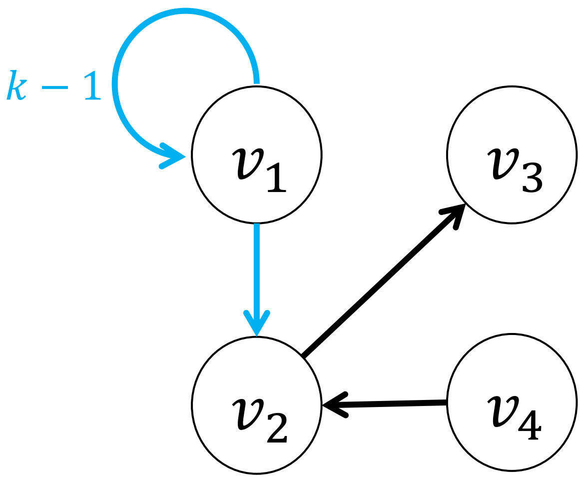
From

$$\mathbf{A}_1^2 = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \underbrace{\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{v_1 \quad v_2 \quad v_3 \quad v_4}$$

To

Common Paths in Graphs

- Number of paths of length k from one node to any other are given by the k^{th} power of the adjacency matrix.



$G_1 = (V_1, E_1)$

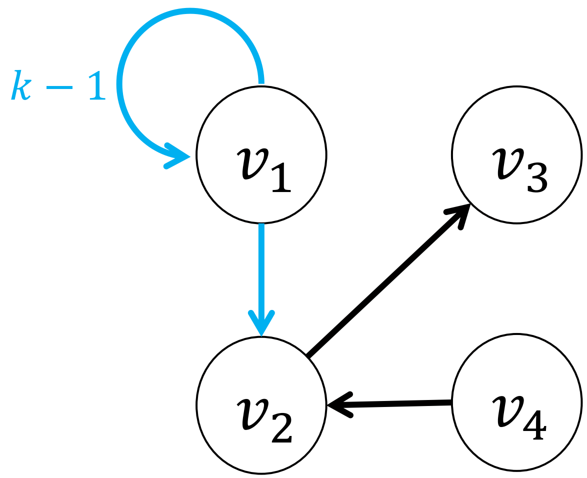
From

$$\mathbf{A}_1^k = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \underbrace{\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}}_{v_1 \quad v_2 \quad v_3 \quad v_4} \quad k > 2$$

To

Common Paths in Graphs

- Number of paths of length k from one node to any other are given by the k^{th} power of the adjacency matrix.



$G_1 = (V_1, E_1)$

From

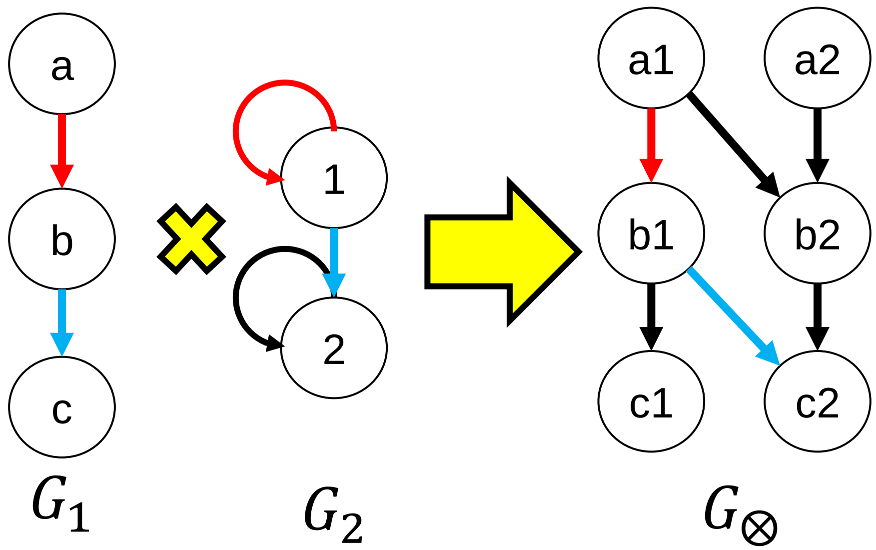
$$\mathbf{A}_1^k = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 1 & 1 & 1 & 0 \\ v_2 & 0 & 0 & 0 & 0 \\ v_3 & 0 & 0 & 0 & 0 \\ v_4 & 0 & 0 & 0 & 0 \end{array} \quad k > 2$$

To

- Number of paths of length k: $\sum_{i,j=1}^{n} \left(\mathbf{A}^k\right)_{ij} = \mathbf{1}^{\mathrm{T}} \mathbf{A}^k \mathbf{1}$

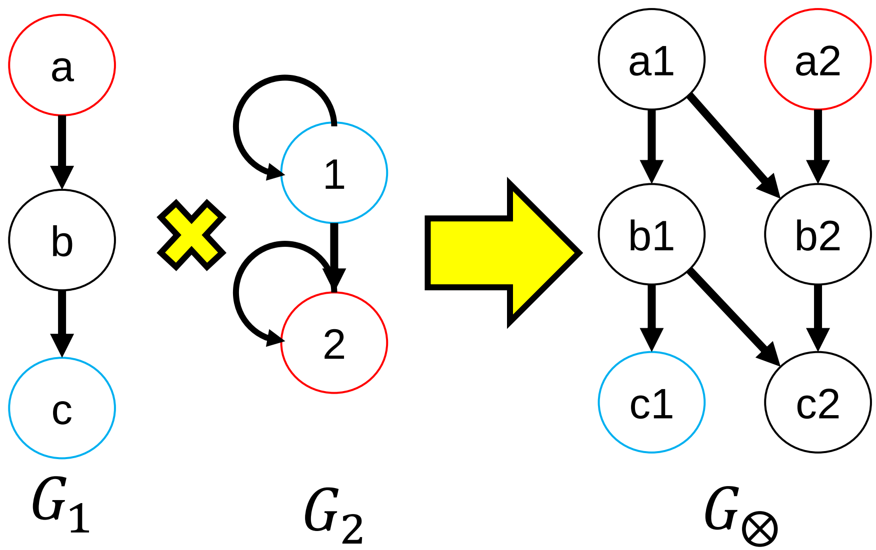
Common Paths in Graphs

- Common paths are given by product graphs $G_\otimes = (V_\otimes, E_\otimes)$:
  - $V_\otimes = V_1 \otimes V_2$
  - $E_\otimes = \{((v, v'), (w, w')) \mid (v, w) \in E_1 \land (v', w') \in E_2\}$



Similarity between Graphs

- Similarity between graphs: number of "common" paths in their product graph.



From

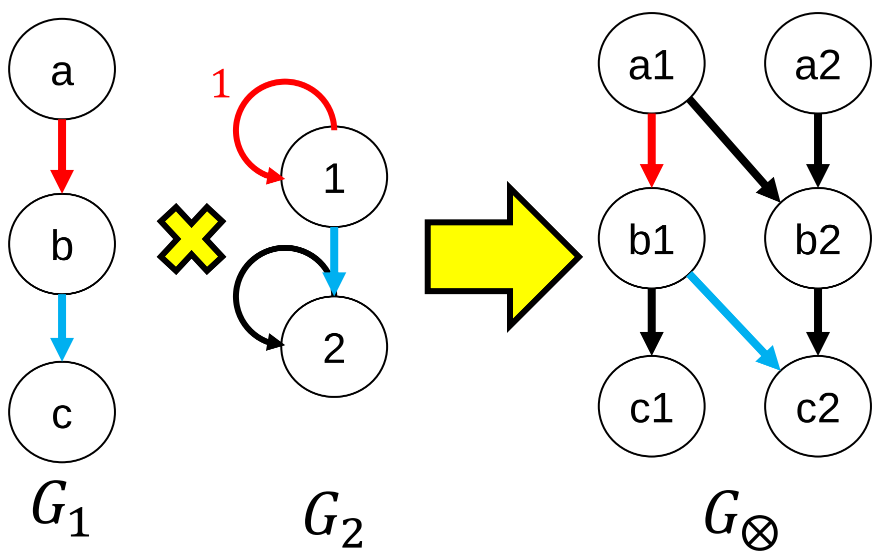
$$\mathbf{A}_\otimes^0 = \begin{array}{c} a1 \\ a2 \\ b1 \\ b2 \\ c1 \\ c2 \end{array} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_{a1 \quad a2 \quad b1 \quad b2 \quad c1 \quad c2}$$

To

$$CP_{\leq 0} = \sum_{i,j=1}^{n} (\mathbf{A}^0)_{ij} = 6$$

Similarity between Graphs

- Similarity between graphs: number of “common” paths in their product graph.



From

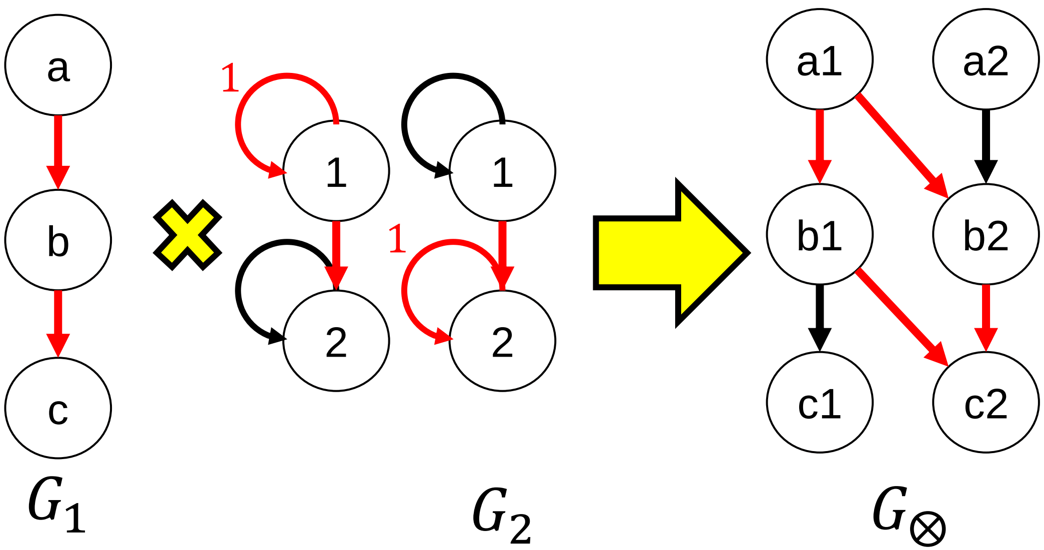
$$\mathbf{A}_{\otimes} = \begin{array}{c} a1 \\ a2 \\ b1 \\ b2 \\ c1 \\ c2 \end{array} \underbrace{\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}_{a1 \quad a2 \quad b1 \quad b2 \quad c1 \quad c2}$$

To

$$CP_{\leq 1} = CP_{\leq 0} + \sum_{i,j=1}^{n} (\mathbf{A}^1)_{ij} = 6 + 6 = 12$$

Similarity between Graphs

- Similarity between graphs: number of “common” paths in their product graph.



From

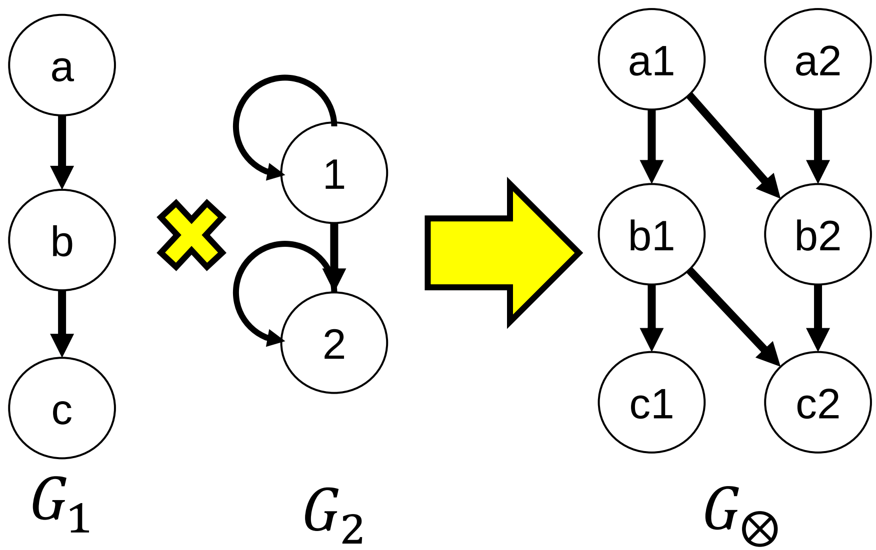
$$\mathbf{A}_\otimes^2 = \begin{array}{c} a1 \\ a2 \\ b1 \\ b2 \\ c1 \\ c2 \end{array} \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}_{a1 \quad a2 \quad b1 \quad b2 \quad c1 \quad c2}$$

To

$$CP_{\leq 2} = CP_{\leq 1} + \sum_{i,j=1}^{n} (\mathbf{A}^2)_{ij} = 12 + 4 = 16$$

Similarity between Graphs

- Similarity between graphs: number of "common" paths in their product graph.



From

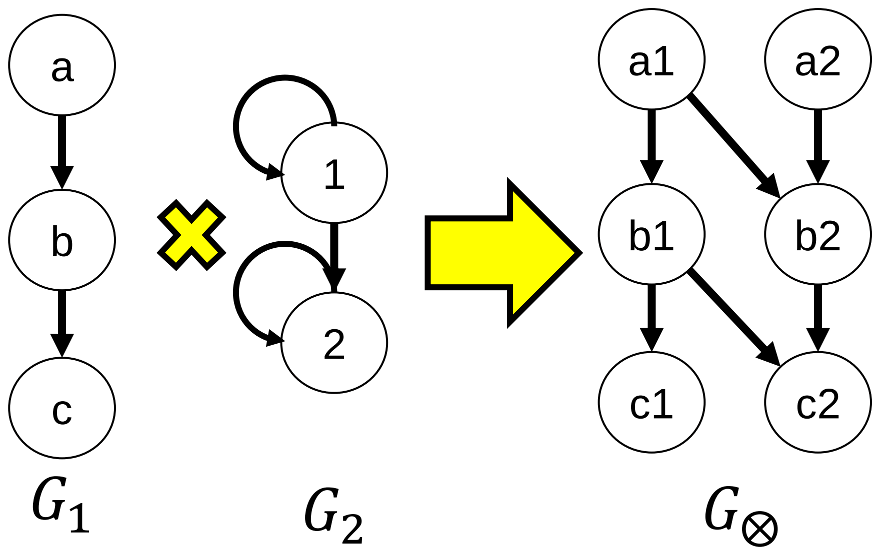
$$
\mathbf{A}_\otimes^3 = \begin{array}{c} a1 \\ a2 \\ b1 \\ b2 \\ c1 \\ c2 \\ \\ \end{array} \begin{array}{c} \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}} \\ \begin{array}{cccccc} a1 & a2 & b1 & b2 & c1 & c2 \end{array} \end{array}
$$

To

$$
CP_{\leq 3} = CP_{\leq 2} + \sum_{i,j=1}^{n} (\mathbf{A}^3)_{ij} = 16 + 0 = 16
$$

Similarity between Graphs

- Similarity between graphs: number of “common” paths in their product graph.



From

$$
\mathbf{A}_\otimes^k = \begin{array}{c} a1 \\ a2 \\ b1 \\ b2 \\ c1 \\ c2 \end{array}
\underbrace{\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}}_{a1 \quad a2 \quad b1 \quad b2 \quad c1 \quad c2}
$$

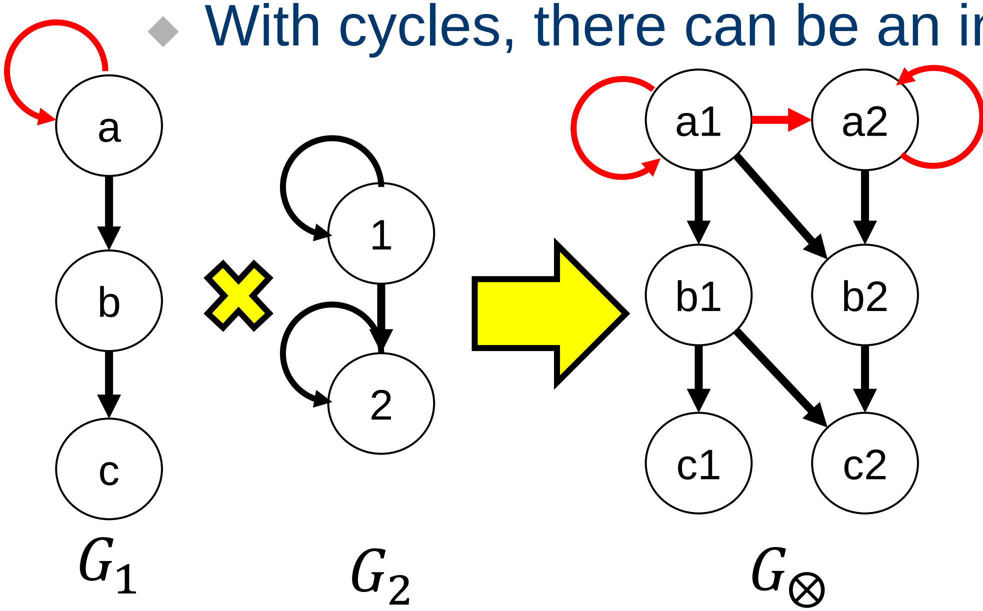
To

$$k > 2$$

$$CP_{\leq\infty} = \sum_{k=0}^{\infty} \sum_{i,j=1}^{n} \left(\mathbf{A}^k\right)_{ij} = 16$$

Similarity between Graphs

- Similarity between graphs: number of “common” paths in their product graph.
  - With cycles, there can be an infinite number paths!



From

$$\mathbf{A}_\otimes^k = \begin{array}{c c} \begin{array}{c} a1 \\ a2 \\ b1 \\ b2 \\ c1 \\ c2 \end{array} & \begin{pmatrix} 1 & k & 1 & k & 1 & k \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\ & \underbrace{\begin{array}{cccccc} & & & & & \end{array}}_{\begin{array}{cccccc} a1 & a2 & b1 & b2 & c1 & c2 \end{array}} \end{array}$$

To

$$k > 2$$

$$CP_{\leq L} = \sum_{k=0}^{L} \sum_{i,j=1}^{n} \left(\mathbf{A}^k\right)_{ij} = \frac{3}{2}L^2 + \frac{15}{2}L + 6 \rightarrow \infty$$

Similarity between Graphs

- Similarity between graphs: number of "common" paths in their product graph.
  - With cycles, there can be an infinite number paths!
  - ⇨ We must downweight the influence of long paths.
- Random Walk Kernels:

$$k(G_1, G_2) = \frac{1}{|V_1||V_2|} \sum_{k=0}^{\infty} \sum_{i,j=1}^{n} \lambda^k \left(\mathbf{A}_\otimes^k\right)_{ij} = \frac{\mathbf{1}^\mathrm{T}(\mathbf{I} - \lambda \mathbf{A}_\otimes)^{-1}\mathbf{1}}{|V_1||V_2|}$$

$$k(G_1, G_2) = \frac{1}{|V_1||V_2|} \sum_{k=0}^{\infty} \sum_{i,j=1}^{n} \frac{\lambda^k}{k!} \left(\mathbf{A}_\otimes^k\right)_{ij} = \frac{\mathbf{1}^\mathrm{T}\exp(\lambda \mathbf{A}_\otimes)\mathbf{1}}{|V_1||V_2|}$$

- These kernels can be calculated by means of the Sylvester Equation in $O(n^3)$.

Similarity between Nodes

- Similarity between graphs: number of “common” paths in their product graph.
- Assumption: Nodes are similar if they are connected by many paths.
- Random Walk Kernels:

$$k(v_i, v_j) = \sum_{k=1}^{\infty} \lambda^k \left(\mathbf{A}_{\otimes}^k\right)_{ij} = \left((\mathbf{I} - \lambda \mathbf{A}_{\otimes})^{-1}\right)_{ij}$$

$$k(v_i, v_j) = \sum_{k=1}^{\infty} \frac{\lambda^k}{k!} \left(\mathbf{A}_{\otimes}^k\right)_{ij} = \left(\exp(\lambda \mathbf{A}_{\otimes})\right)_{ij}$$

Additional Graph-Kernels

- Shortest-Path Kernel
  - All shortest paths between pairs of nodes computed by Floyd-Warshall algorithm with run time $O(|V|^3)$
  - Compare all pairs of shortest paths between 2 graphs $O(|V_1|^2 |V_2|^2)$

- Subtree-Kernel:
  - Idea: use tree structures as indexes in the feature space
  - Can be recursively computed for a fixed height tree
  - Trees are downweighted in their height

Summary

- Kernel function $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{x}')$ computes the inner product of the feature mapping of instances.
- The kernel function can often be computed without an explicit representation $\phi(\mathbf{x})$.
  - E.g., polynomial kernel: $k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1)^p$
- Infinite-dimensional feature mappings are possible
  - Eg., RBF kernel: $k_{RBF}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2}$
- Kernel functions for time series, strings, graphs, …
- For a given kernel matrix, the Mercer map provides a feature mapping.

Summary

- Representer Theorem: $f_{\boldsymbol{\theta}^*}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x})$
  - Instances only interact through inner products
  - Great for few instances, many attributes
- Kernel learning algorithms:
  - Kernel ridge regression
  - Kernel perceptron, SVM,
  - …