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: \( f_\theta(x) = x^T \theta \)

- Regularized empirical risk minimization:
  \[
  \arg\min_\theta \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) + \lambda \Omega(\theta)
  \]

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

- Models constrained to hyperplane in feature space: \( H_\theta = \{x|x^T\theta = 0\} \).
- Use mapping \( \phi \) to embed instances \( x \in X \) 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 $\phi$.

$\phi(x_1, x_2) = (x_1 x_2, x_1^2, x_2^2)$

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

- **Instances:**
  \[
  X = \begin{pmatrix}
  x_{11} & \cdots & x_{1m} \\
  \vdots & \ddots & \vdots \\
  x_{n1} & \cdots & x_{nm}
  \end{pmatrix}
  \]

- **Feature Mapping:**
  \[
  \Phi = \begin{pmatrix}
  \phi(x_1)^T \\
  \vdots \\
  \phi(x_n)^T
  \end{pmatrix}
  = \begin{pmatrix}
  \phi(x_1)_1 & \cdots & \phi(x_1)_{m'} \\
  \vdots & \ddots & \vdots \\
  \phi(x_n)_1 & \cdots & \phi(x_n)_{m'}
  \end{pmatrix}
  \]
Feature Mappings

- Feature mapping $\phi(x)$ can be high dimensional.
  - The size of estimated parameter vector $\theta$ depends on the dimensionality of $\phi$ – could be infinite!

- Computation of $\phi(x)$ can be expensive.
  - $\phi$ must be computed for each training point $x_i$ & for each prediction $x$.

- How can we adapt linear methods to efficiently incorporate high dimensional $\phi$?
Representer Theorem: Observation

- Perceptron algorithm:

- Resulting parameter vector is a linear combination of instances: \( \theta^* = \sum_{i=1}^{n} \alpha_i y_i x_i \)

- Sufficient to determine coefficients \( \alpha_i \), 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 $\theta^*$ that minimizes

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

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

$$f_{\theta^*}(x) = \sum_{i=1}^{n} \alpha_i^* \phi(x_i)^T \phi(x)$$

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

Inner product is a measure for similarity between instances.
Representer Theorem: Proof

■ Orthogonal Decomposition:

\[ \theta^* = \theta_{\parallel} + \theta_{\perp}, \text{ with } \theta_{\parallel} \in \Theta_{\parallel} = \{ \sum_{i=1}^{n} \alpha_i \phi(x_i) | \alpha_i \in \mathbb{R} \} \]

and \( \theta_{\perp} \in \Theta_{\perp} = \{ \theta \in \Theta | \theta^T \theta_{\parallel} = 0 \ \forall \ \theta_{\parallel} \in \Theta_{\parallel} \} \]

\[ L(\theta) = \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) + g(\|f_\theta\|_2) \]
Intelligent Data Analysis

**Representer Theorem: Proof**

- **Orthogonal Decomposition:**
  - $\theta^* = \theta_\parallel + \theta_\perp$, with $\theta_\parallel \in \Theta_\parallel = \{\sum_{i=1}^{n} \alpha_i \phi(x_i) | \alpha_i \in \mathbb{R}\}$
  - and $\theta_\perp \in \Theta_\perp = \{\theta \in \Theta | \theta^T \theta_\parallel = 0 \ \forall \ \theta_\parallel \in \Theta_\parallel\}$

- For any training point $x_i$ it follows that
  $$f_{\theta^*}(x_i) = \theta_\parallel^T \phi(x_i) + \theta_\perp^T \phi(x_i) = \theta_\parallel^T \phi(x_i)$$
  - Why is $\theta_\perp^T \phi(x_i) = 0$?

\[
L(\theta) = \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) + g(\|f_\theta\|_2)
\]
Representer Theorem: Proof

- **Orthogonal Decomposition:**
  - $\theta^* = \theta_\parallel + \theta_\perp$, with $\theta_\parallel \in \Theta_\parallel = \{\sum_{i=1}^{n} \alpha_i \phi(x_i) \mid \alpha_i \in \mathbb{R}\}$ and $\theta_\perp \in \Theta_\perp = \{\theta \in \Theta \mid \theta^T \theta_\parallel = 0 \; \forall \; \theta_\parallel \in \Theta_\parallel\}$
  - For any training point $x_i$, it follows that $f_{\theta^*}(x_i) = \theta_\parallel^T \phi(x_i) + \theta_\perp^T \phi(x_i) = \theta_\parallel^T \phi(x_i)$
  - $\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$ is independent of $\theta_\perp$.
  - because $\theta_\perp^T \phi(x_i) = 0$

- **Finally from** $g(\|\theta^*\|_2) \geq g(\|\theta_\parallel\|_2)$, it follows $\theta_\perp = 0$.

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

Since $\theta_\perp^T \theta_\parallel = 0$ (Pythagoras' Theorem)

Since $g$ is strictly monotonically increasing.
Representer Theorem

- The hyperplane $\theta^*$, which minimizes
  - $L(\theta) = \sum_{i=1}^{n} \ell(\theta^T \phi(x), y_i) + \Omega(\theta)$
- can be represented as
  $$f_{\theta^*}(x) = \theta^* T \phi(x) = f_{\alpha^*}(x) = \sum_{i=1}^{n} \alpha_i^* \phi(x_i)^T \phi(x)$$
Primal vs. Dual View

- Primal decision function:
  \[ f_\theta(x) = \theta^T \phi(x) \]

- Dual decision function:
  \[ f_\alpha(x) = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \phi(x) \]
**Primal vs. Dual View**

- Primal decision function:
  \[ f_\theta(x) = \theta^T \phi(x) \]

- Dual decision function:
  \[ f_\alpha(x) = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \phi(x) = \alpha^T \Phi \phi(x) \]

- Illustration:
  \[
  \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \\
  = (\alpha_1 \ldots \alpha_n) \\
  \begin{pmatrix}
  - & \phi(x_1)^T & - \\
  \vdots & & \vdots \\
  - & \phi(x_n)^T & -
  \end{pmatrix} = \alpha^T \Phi
  \]
Primal vs. Dual View

- Primal decision function:
  \[ f_\theta(x) = \theta^T \phi(x) \]

- Dual decision function:
  \[ f_\alpha(x) = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \phi(x) = \alpha^T \Phi \phi(x) \]

- Duality between parameters:
  \[ \theta = \sum_{i=1}^{n} \alpha_i \phi(x_i) = \Phi^T \alpha \]

- Illustration:
  \[
  \theta = \left(\begin{array}{c|c|c}
  \phi(x_1) & \cdots & \phi(x_n)
  \end{array}\right) \left(\begin{array}{c}
  \alpha_1 \\
  \vdots \\
  \alpha_n
  \end{array}\right) = \Phi^T \alpha
  \]
Primal vs. Dual View

- Primal decision function:
  \[ f_\theta(x) = \theta^T \phi(x) \]

- Dual decision function:
  \[ f_\alpha(x) = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \phi(x) = \alpha^T \Phi \phi(x) \]

- Duality between parameters:
  \[ \theta = \sum_{i=1}^{n} \alpha_i \phi(x_i) = \Phi^T \alpha \]
Primal vs. Dual View

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

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

- Dual view of the decision function:

\[
    f_\alpha(x) = \left( \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \right) \phi(x)
    = \sum_{i=1}^{n} \alpha_i \left( \phi(x_i)^T \phi(x) \right)
    = \sum_{i=1}^{n} \alpha_i k(x_i, x)
\]

- Where kernel function \( k(x_i, x) \) calculates the inner product \( \phi(x_i)^T \phi(x) \).
Kernel Functions

- Kernel functions can be understood as a measure of similarity between instances.
- Primal view on data: “what does \( \mathbf{x} \) look like?”
  \[
  \phi(\mathbf{x}) = \begin{pmatrix} \phi(x)_1 \\ \vdots \\ \phi(x)_{m'} \end{pmatrix} \Rightarrow \text{multiply by } \theta^T.
  \]
- Dual view on data: “how similar is \( \mathbf{x} \) to each training instance?”
  \[
  \mathbf{\Phi} \phi(\mathbf{x}) = \begin{pmatrix} k(x_1, \mathbf{x}) \\ \vdots \\ k(x_n, \mathbf{x}) \end{pmatrix} \Rightarrow \text{multiply by } \alpha^T.
  \]
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:
  \[ \ell_2(f_\theta(x_i), y_i) = (f_\theta(x_i) - y_i)^2 \]

- L2 regularization:
  \[ \Omega_2(\theta) = \|\theta\|_2^2 \]
Kernel Ridge Regression

Minimize

\[ L(\theta) = \sum_{i=1}^{n} (f_{\theta}(x_i) - y_i)^2 + \lambda \theta^T \theta \]

\[ = \sum_{i=1}^{n} (\theta^T \phi(x_i) - y_i)^2 + \lambda \theta^T \theta \]
Kernel Ridge Regression

- Minimize

\[
L(\theta) = \sum_{i=1}^{n} (f_{\theta}(x_i) - y_i)^2 + \lambda \theta^T \theta
\]

\[
= \sum_{i=1}^{n} (\theta^T \phi(x_i) - y_i)^2 + \lambda \theta^T \theta
\]

\[
= (\Phi \theta - y)^T (\Phi \theta - y) + \lambda \theta^T \theta
\]

- Why?

\[
(\Phi \theta - y) = \begin{pmatrix}
- & \phi(x_1)^T \\
& \vdots \\
- & \phi(x_n)^T \\
\end{pmatrix}
\begin{pmatrix}
\theta_1 \\
\vdots \\
\theta_m \\
\end{pmatrix}
- y
\]

\[
= \begin{pmatrix}
\phi(x_1)^T \theta - y_1 \\
\vdots \\
\phi(x_n)^T \theta - y_n \\
\end{pmatrix}
\]
Kernel Ridge Regression

- Minimize

\[ L(\theta) = \sum_{i=1}^{n} (f_\theta(x_i) - y_i)^2 + \lambda \theta^T \theta \]

\[ = \sum_{i=1}^{n} (\theta^T \phi(x_i) - y_i)^2 + \lambda \theta^T \theta \]

\[ = (\Phi \theta - y)^T (\Phi \theta - y) + \lambda \theta^T \theta \]

- By the representer theorem:

\[ \theta = \Phi^T \alpha \]

- Dual regularized empirical risk:

\[ L(\alpha) = (\Phi \Phi^T \alpha - y)^T (\Phi \Phi^T \alpha - y) + \lambda \alpha^T \Phi \Phi^T \alpha \]
Kernel Ridge Regression

- Dual regularized empirical risk:
  \[
  L(\alpha) = (\Phi \Phi^T \alpha - y)^T(\Phi \Phi^T \alpha - y) + \lambda \alpha^T \Phi \Phi^T \alpha \\
  = \alpha^T \Phi^T \Phi \Phi^T \alpha - 2 \alpha^T \Phi \Phi^T y - y^T y \\
  + \lambda \alpha^T \Phi \Phi^T \alpha
  \]

- Define gram matrix (or kernel matrix) as \( K = \Phi \Phi^T \).
  \[
  L(\alpha) = \alpha^T KK\alpha - 2 \alpha^T Ky - y^T y + \lambda \alpha^T K\alpha
  \]

- Setting the derivative to zero
  \[
  \frac{\partial}{\partial \alpha} L(\alpha) = 0
  \]

- Gives the solution
  \[
  \alpha = (K + \lambda I)^{-1}y
  \]
Kernel Ridge Regression

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

$$\mathbf{K} = \begin{pmatrix} - & \phi(\mathbf{x}_1)^T & - \\ \vdots & \ddots & \vdots \\ - & \phi(\mathbf{x}_n)^T & - \end{pmatrix} \begin{pmatrix} \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \end{pmatrix} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

- $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$
Kernel Ridge Regression

- Regression method that uses kernel functions
- Works with any nonlinear embedding $\phi$ as long as there is a kernel function that computes the inner product: $k(x_i, x) = \phi(x_i)^T \phi(x)$.
- Kernel matrix $K$ of size $n \times n$ has to be inverted, works only for modest sample sizes.
- Solution dependent on $K_{ij} = k(x_i, x_j)$, but otherwise independent of $\Phi$.
- 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:
  \[ \ell_p(f_\theta(x_i), y_i) = \max(0, -y_i f_\theta(x_i)) \]
- No regularizer.
- Primal stochastic gradient:
  \[ \nabla L_{x_i}(\theta) = \begin{cases} 
  -y_i x_i & -y_i f_\theta(x_i) > 0 \\
  0 & -y_i f_\theta(x_i) < 0 
\end{cases} \]

Rosenblatt, 1960
Kernel Perceptron

- Stochastic gradient update step:

\[ \theta' = \theta + y_i \phi(x_i) \]

\[ \iff \sum_{i=1}^{n} \alpha'_i \phi(x_i)^T = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T + y_i \phi(x_i) \]

\[ \iff \alpha'_i = \alpha_i + y_i \]

- Dual stochastic gradient update step:

\[ \text{IF } y_i f_\theta(x_i) \leq 0 \]
\[ \text{THEN } \theta' = \theta + y_i x_i \]

\[ \text{IF } y_i f_\alpha(x_i) \leq 0 \]
\[ \text{THEN } \alpha_i = \alpha_i + y_i \]
Kernel Perceptron Algorithm

Perceptron(Instances \{ (x_i, y_i) \})

Set $\alpha = 0$

DO

FOR $i = 1, \ldots, n$

IF $y_i f_\alpha(x_i) \leq 0$

THEN $\alpha_i = \alpha_i + y_i$

END

WHILE $\alpha$ changes

RETURN $\alpha$

- Decision function:

$$f_\alpha(x) = \alpha^T \Phi \phi(x_i) = \sum_{i=1}^{n} \alpha_i k(x_i, x)$$
Kernel Perceptron

- Perceptron loss, no regularizer
- Dual form of the decision function:
  \[ f_\alpha(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) \]
- Dual form of the update rule:
  - If \( y_i f_\alpha(x_i) \leq 0 \), 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 \( \phi(x) \) 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: $\min_\theta \left[ \sum_{i=1}^{n} \max(0, 1 - y_i \phi(x_i)^T \theta) + \frac{1}{2\lambda} \theta^T \theta \right]$

- Equivalent optimization problem with side constraints:

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

  such that

  $$y_i \phi(x_i)^T \theta \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

- Goal: dual formulation of the optimization problem
Kernel Support Vector Machine

- Optimization problem with side constraints:
  \[
  \min_{\theta, \xi} \left[ \lambda \sum_{i=1}^{n} \xi_i + \frac{1}{2} \theta^T \theta \right] \\
  \text{such that} \\
  y_i \phi(x_i)^T \theta \geq 1 - \xi_i \text{ and } \xi_i \geq 0
  \]

- Lagrange function with Lagrange-Multipliers \( \beta \geq 0 \) and \( \beta^0 \geq 0 \) for the side constraints:
  \[
  L(\theta, \xi, \beta, \beta^0) = \lambda \sum_{i=1}^{n} \xi_i + \frac{\theta^T \theta}{2} - \sum_{i=1}^{n} \beta_i (y_i \phi(x_i)^T \theta - 1 + \xi_i) - \sum_{i=1}^{n} \beta_i^0 \xi_i
  \]

- Optimization problem without side constraints:
  \[
  \min_{\theta, \xi} \max_{\beta, \beta^0} L(\theta, \xi, \beta, \beta^0)
  \]
Kernel Support Vector Machine

- Lagrange function:
  \[ L(\theta, \xi, \beta, \beta^0) = \lambda \sum_{i=1}^{n} \xi_i + \frac{\theta^T \theta}{2} - \sum_{i=1}^{n} \beta_i (y_i \phi(x_i)^T \theta - 1 + \xi_i) - \sum_{i=1}^{n} \beta_i^0 \xi_i \]

- Setting the derivative of \( L \) w.r.t. \( (\theta, \xi) \) to zero gives:
  \[ \frac{\partial}{\partial \theta} L(\theta, \xi, \beta, \beta^0) = 0 \Rightarrow \theta = \sum_{i=1}^{n} \beta_i y_i \phi(x_i) \]
  \[ \frac{\partial}{\partial \xi_i} L(\theta, \xi, \beta, \beta^0) = 0 \Rightarrow \lambda = \beta_i + \beta_i^0 \]

Relation between primal and dual parameters… representer theorem.
Kernel Support Vector Machine

- Substitute the derived parameters into the Lagrange function:

\[
L(\theta, \xi, \beta, \beta^0) = \frac{1}{2} \theta^T \theta
\]

\[
- \sum_{i=1}^{n} \beta_i (y_i \phi(x_i)^T \theta - 1 + \xi_i) - \sum_{i=1}^{n} \beta_i^0 \xi_i + \lambda \sum_{i=1}^{n} \xi_i
\]
Substitute the derived parameters into the Lagrange function:

\[
L(\theta, \xi, \beta, \beta^0) = \frac{1}{2} \left( \sum_{i=1}^{n} \beta_i y_i \phi(x_i) \right)^T \left( \sum_{j=1}^{n} \beta_j y_j \phi(x_j) \right) \\
- \sum_{i=1}^{n} \beta_i \left( y_i \phi(x_i)^T \sum_{j=1}^{n} \beta_j y_j \phi(x_j) - 1 + \xi_i \right) \\
- \sum_{i=1}^{n} \beta_i^0 \xi_i + \lambda \sum_{i=1}^{n} \xi_i
\]

\[
\theta = \sum_{i=1}^{n} \beta_i y_i \phi(x_i) \\
\lambda = \beta_i + \beta_i^0
\]
Kernel Support Vector Machine

- Substitute the derived parameters into the Lagrange function:

\[
L(\theta, \xi, \beta, \beta^0) = \frac{1}{2} \left( \sum_{i=1}^{n} \beta_i y_i \phi(x_i) \right)^T \left( \sum_{j=1}^{n} \beta_j y_j \phi(x_j) \right)
- \sum_{i=1}^{n} \beta_i \left( y_i \phi(x_i)^T \sum_{j=1}^{n} \beta_j y_j \phi(x_j) - 1 + \xi \right) - \sum_{i=1}^{n} \beta_i^0 \xi_i + \lambda \sum_{i=1}^{n} \xi_i
= \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(x_i)^T \phi(x_j)
- \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(x_i)^T \phi(x_j) + \sum_{i=1}^{n} \beta_i - \sum_{i=1}^{n} \left( \beta_i + \beta_i^0 \right) \xi_i + \lambda \sum_{i=1}^{n} \xi_i
\]
Kernel Support Vector Machine

- Substitute the derived parameters into the Lagrange function:

\[
L(\theta, \xi, \beta, \beta^0) = \frac{1}{2} \left( \sum_{i=1}^{n} \beta_i y_i \phi(x_i) \right)^T \left( \sum_{j=1}^{n} \beta_j y_j \phi(x_j) \right) \\
- \sum_{i=1}^{n} \beta_i \left( y_i \phi(x_i)^T \sum_{j=1}^{n} \beta_j y_j \phi(x_j) - 1 + \xi_i \right) - \sum_{i=1}^{n} \beta^0_i \xi_i + \lambda \sum_{i=1}^{n} \xi_i \\
= \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(x_i)^T \phi(x_j) \\
- \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(x_i)^T \phi(x_j) + \sum_{i=1}^{n} \beta_i - \sum_{i=1}^{n} \left( \beta_i + \beta^0_i \right) \xi_i + \lambda \sum_{i=1}^{n} \xi_i \\
= \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \phi(x_i)^T \phi(x_j)
\]

\[
\theta = \sum_{i=1}^{n} \beta_i y_i \phi(x_i) \\
\lambda = \beta_i + \beta^0_i
\]
Kernel Support Vector Machine

- Optimization criterion of the dual SVM:

\[
\max_{\beta} \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j k(x_i,x_j)
\]

such that

\[0 \leq \beta_i \leq \lambda\]

L1-Regularizer of \(\beta\) (sparse)

Large if \(\beta_i, \beta_j > 0\) for similar instances of different classes.
Kernel Support Vector Machine

- Optimization criterion of the dual SVM:
  \[
  \max_{\beta} \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j k(x_i, x_j)
  \]

- Optimization over parameters $\beta$.
- 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.
  \[ \theta = \sum_{x_i \in SV} \beta_i y_i \phi(x_i) \]

- Dual form of the decision function:
  \[ f_\beta(x) = \sum_{x_i \in SV} \beta_i y_i k(x_i, x) \]

- Primal SVM:
  ◆ Solution is a Vector \( \theta \) in the space of the attributes.

- Dual SVM:
  ◆ The same solution is represented as weights \( \beta_i \) of the samples.

Support Vectors: \( \beta_i > 0 \)
Constructing Kernels

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

- Kernel matrices are symmetric:
  \[ \mathbf{K} = \mathbf{K}^T \]
- Kernel matrices \( \mathbf{K} \in \mathbb{R}^{n \times n} \) are positive semidefinite:
  \[ \exists \mathbf{\Phi} \in \mathbb{R}^{n \times m} : \mathbf{K} = \mathbf{\Phi} \mathbf{\Phi}^T \]
- 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 \( k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') \) 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
Eigenvalue decomposition: Every symmetric matrix $\mathbf{K}$ can be decomposed in terms of its eigenvectors $\mathbf{u}_i$ and eigenvalues $\lambda_i$:

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

If $\mathbf{K}$ is positive semi-definite, then $\lambda_i \in \mathbb{R}^{0+}$.

The eigenvectors are orthonormal ($\mathbf{u}_i^T \mathbf{u}_i = 1$ and $\mathbf{u}_i^T \mathbf{u}_j = 0$) and $\mathbf{U}$ is orthogonal: $\mathbf{U}^T = \mathbf{U}^{-1}$. 
Mercer Map

- Thus it holds:
  \[ K = \mathbf{U}\Lambda\mathbf{U}^T \]
  \[ = (\mathbf{U}\Lambda^{1/2})(\Lambda^{1/2}\mathbf{U}^T) \]
  \[ = (\mathbf{U}\Lambda^{1/2})(\mathbf{U}\Lambda^{1/2})^T \]

- Feature mapping for training data can be defined as
  \[
  \begin{pmatrix}
  \phi(x_1) & \cdots & \phi(x_n)
  \end{pmatrix}
  = (\mathbf{U}\Lambda^{1/2})^T
  
  \text{Eigenvalue decomposition}

  \text{Diagonal matrix with } \sqrt{\lambda_i}
Mercer Map

- Feature mapping for used training data can then be defined as
  \[
  \begin{pmatrix}
  \phi(x_1) \\ \vdots \\ \phi(x_n)
  \end{pmatrix}
  = (U\Lambda^{1/2})^T
  \]

- Kernel matrix between training and test data
  \[
  K_{test} = \Phi(X_{train})^T\Phi(X_{test})
  = (U\Lambda^{1/2})\Phi(X_{test})
  \]

- Equation results in a mapping of the test data:
  \[
  \Phi(X_{test}) = (U\Lambda^{1/2})^{-1}K_{test}
  \]
  \[
  \Phi(X_{test}) = \Lambda^{-1/2}U^TK_{test}
  \]
  \[
  U^T = U^{-1}
  \]
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:
  \[ k(x, x') = ck_1(x, x') \]
  \[ k(x, x') = f(x)k_1(x, x')f(x') \]
  \[ k(x, x') = q(k_1(x, x')) \]
  \[ k(x, x') = e^{k_1(xx')} \]
  \[ k(x, x') = k_1(x, x') + k_2(x, x') \]
  ...

Intelligent Data Analysis

Kernel Compositions
Kernel Functions

- Polynomial kernels: \( k_{poly}(x_i, x_j) = (x_i^T x_j + 1)^p \)
- Radial basis functions: \( k_{RBF}(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2} \)
- Sigmoid kernels,
- Dynamic time-warping kernels,
- String kernels,
- Graph kernels,
Polynomial Kernels

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

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

$$= \left( (x_{i1} \quad x_{i2}) \left( x_{j1} \quad x_{j2} \right) + 1 \right)^2 = (x_{i1}x_{j1} + x_{i2}x_{j2} + 1)^2$$
Polynomial Kernels

- Kernel: $k_{poly}(x_i, x_j) = (x_i^T x_j + 1)^p$, 2D-input, $p = 2$.

$$
k_{poly}(x_i, x_j) = (x_i^T x_j + 1)^2
= \left( (x_{i1} \quad x_{i2}) \left( \begin{array}{c} x_{j1} \\ x_{j2} \end{array} \right) + 1 \right)^2
= (x_{i1}x_{j1} + x_{i2}x_{j2} + 1)^2
= (x_{i1}^2x_{j1} + x_{i2}^2x_{j2} + 2x_{i1}x_{j1}x_{i2}x_{j2} + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 1)
$$

$$
= (x_{i1}^2 \quad x_{i2}^2 \quad \sqrt{2}x_{i1}x_{i2} \quad \sqrt{2}x_{i1} \quad \sqrt{2}x_{i2} \quad 1)
\begin{pmatrix}
\phi(x_i)^T \\
\phi(x_j)
\end{pmatrix}
$$

All monomials of degree ≤2 over input attributes
Polynomial Kernels

- Kernel: \( k_{\text{poly}}(x_i, x_j) = (x_i^T x_j + 1)^p \), 2D-input, \( p = 2 \).

\[
k_{\text{poly}}(x_i, x_j) = (x_i^T x_j + 1)^2
\]

\[
= \left( \begin{pmatrix} x_{i1} & x_{i2} \end{pmatrix} \begin{pmatrix} x_{j1} \\ x_{j2} \end{pmatrix} + 1 \right)^2
= \left( x_{i1}x_{j1} + x_{i2}x_{j2} + 1 \right)^2
\]

\[
= (x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 1)
\]

\[
= \left( \begin{pmatrix} x_{i1} \\ x_{i2} \end{pmatrix} \begin{pmatrix} \sqrt{2}x_{i1} \\ \sqrt{2}x_{i2} \end{pmatrix} \begin{pmatrix} \sqrt{2}x_{j1} \\ \sqrt{2}x_{j2} \end{pmatrix} \begin{pmatrix} 1 \end{pmatrix} \right)_{\phi(x_i)^T}
\]

All monomials of degree \( \leq 2 \) over input attributes

\[
= \left( \begin{pmatrix} x_i \otimes x_i \\ \sqrt{2}x_i \\ 1 \end{pmatrix} \right)^T \left( \begin{pmatrix} x_j \otimes x_j \\ \sqrt{2}x_j \\ 1 \end{pmatrix} \right)
\]
RBF Kernel

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

- Similarity of time series
- Idea: Find corresponding similar points in \( x, x' \).
- Correspondence function
  \[ \pi_x(k) \in [1, T_x], \pi_{x'}(l) \in [1, T_{x'}] \]
- DTW distance is squared distance between matched sequences:
  \[
k_{DTW}(x, x') = e^{-\left( \min \sum_{k=1}^{T} \left( x_{\pi_x(k)} - x'_{\pi_{x'}(k)} \right)^2 \right)}\]
Intelligent Data Analysis

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:
  $$\gamma(k, l) = (x_k - x_l)^2 + \min\{\gamma(k-1, l-1), \gamma(k-1, l), \gamma(k, l-1)\}$$
- Algorithm:

  ```
  DTW(Sequences x and x')
  Let $\gamma(0,0) = 0; \gamma(k, 0) = \infty; \gamma(0, l) = \infty$
  FOR $k = 1...T_x$
    FOR $l = 1...T_y$
      $\gamma(k, l) = (x_k - x_l)^2 + \min\{\gamma(k-1, l-1), \gamma(k-1, l), \gamma(k, l-1)\}$
  RETURN $\gamma(T_x, T_y)$
  ```
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-acter 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 $\Sigma$ written as $s = s_1s_2 \ldots s_n$ with $|s| = n$.
  - The set of all strings is $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$
  - $s_{i:j} = s_is_{i+1} \ldots s_j$
  - Subsequence: for any $i \in \{0,1\}^n$, $s[i]$ is the elements of $s$ corresponding to elements of $i$ that are 1
    - Eg. If $s=“abcd”$ $s[(1,0,0,1)]=“ad”$

- 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?

\[ m \approx 1,000,000 \]

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 $\Sigma$.

$\phi(s)_u$ is # of times $u \in \Sigma^p$ is contained in string $s$

The $p$-spectrum kernel is the result

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

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>aa</th>
<th>ab</th>
<th>ba</th>
<th>bb</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaab</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bbab</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>aaaa</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>baab</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>aaab</th>
<th>bbab</th>
<th>aaaa</th>
<th>baab</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaab</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>bbab</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>aaaa</td>
<td>6</td>
<td>0</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>baab</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Without explicitly computing this feature map, the $p$-spectrum kernel can be computed as

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

- This computation is $O(|s||t|)$. 
- Using trie data structures, this computation can be reduced to $O(p \cdot \max(|s|, |t|))$. 
- 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 be 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: $A = (a_{ij})_{i,j=1}^{n}$, $a_{ij} = 1$ if $(v_i, v_j) \in E$
  - Adjacency list
  - Incidence matrix

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

$$A_1 = \begin{pmatrix}
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0
\end{pmatrix}$$
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 $f : V_1 \to V_2$ so that

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

$G_1 = (V_1, E_1)$

$G_2 = (V_2, E_2)$
Isomorphisms of Graphs

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

Subgraph isomorphism: NP-hard!
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.

\[ G_1 = (V_1, E_1) \]
Common Paths in Graphs

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

\[ A_1 = \begin{pmatrix}
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{pmatrix} \]

\[ G_1 = (V_1, E_1) \]
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.

$G_1 = (V_1, E_1)$

$$A_1^2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$
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.

$G_1 = (V_1, E_1)$

$A_1^k = \begin{pmatrix}
1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}$

$k > 2$

Proof?
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.

\[ A^{k} \]

- Number of paths of length $k$: \[ \sum_{i,j=1}^{n}(A^{k})_{ij} = 1^{T}A^{k}1 \]
Common Paths in Graphs

- Common paths are given by product graphs $G \otimes = (V \otimes, E \otimes)$:
  - $V \otimes = V_1 \otimes V_2$
  - $E \otimes = \{((v, v'), (w, w')) | (v, w) \in E_1 \land (v', w') \in E_2 \}$
Similarity between Graphs

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

\[
\begin{align*}
G_1 &\,\times\, G_2 = G_\otimes \\
\begin{pmatrix}
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
c_1 & c_2 \\
\end{pmatrix}
\end{align*}
\]

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{pmatrix} = A_\otimes^0
\]

\[
CP_{\leq 0} = \sum_{i,j=1}^{n} (A_\otimes^0)_{ij} = 6
\]
Similarity between Graphs

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

\[ G_1 \times G_2 = G \]
Similarity between Graphs

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

\[ GP_{\leq 2} = GP_{\leq 1} + \sum_{i,j=1}^{n} (A^2)_{ij} = 12 + 4 = 16 \]
Similarity between Graphs

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

\[ G_1 \quad G_2 \quad G_{\otimes} \]

\[ A^3 = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \]

\[ CP_{\leq 3} = CP_{\leq 2} + \sum_{i,j=1}^{n} (A^3)_{ij} = 16 + 0 = 16 \]
Similarity between Graphs

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

\[ G_1 \rightarrow G_2 \rightarrow G_{\otimes} \]

\[ A^k = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \]

\[ CP_{\leq \infty} = \sum_{k=0}^{\infty} \sum_{i,j=1}^{n} (A^k)_{ij} = 16 \]
Similarity between Graphs

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

\[ \text{From} \quad \begin{pmatrix} 1 & k & 1 & k & 1 & k \\ a1 & a2 & b1 & b2 & c1 & c2 \end{pmatrix} \quad \text{To} \quad \begin{pmatrix} 1 & k & 1 & k & 1 & k \\ a1 & a2 & b1 & b2 & c1 & c2 \end{pmatrix} \]

\[ A^k = \begin{pmatrix} 1 & k & 1 & k & 1 & k \\ a1 & a2 & b1 & b2 & c1 & c2 \end{pmatrix} \]

\[ CP_{\leq L} = \sum_{k=0}^{L} \sum_{i,j=1}^{n} (A^k)_{ij} = \frac{3}{2}L^2 + \frac{15}{2}L + 6 \to \infty \]
Similarity between Graphs

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

- Random Walk Kernels:
  \[
  k(G_1, G_2) = \frac{1}{|V_1||V_2|} \sum_{k=0}^{\infty} \sum_{i,j=1}^{n} \lambda^k (A^k_{\otimes})_{ij} = \frac{1^T (I - \lambda A_{\otimes})^{-1} 1}{|V_1||V_2|}
  \]
  \[
  k(G_1, G_2) = \frac{1}{|V_1||V_2|} \sum_{k=0}^{\infty} \sum_{i,j=1}^{n} \frac{\lambda^k}{k!} (A^k_{\otimes})_{ij} = \frac{1^T \exp(\lambda A_{\otimes}) 1}{|V_1||V_2|}
  \]

- 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:

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

\[
k(v_i, v_j) = \sum_{k=1}^{\infty} \frac{\lambda^k}{k!} (A^k)_{ij} = \left(\exp(\lambda A_{\otimes})\right)_{ij}
\]
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 $k(x, x') = \phi(x)^T\phi(x')$ computes the inner product of the feature mapping of instances.
- The kernel function can often be computed without an explicit representation $\phi(x)$.
  - E.g., polynomial kernel: $k_{poly}(x_i, x_j) = (x_i^T x_j + 1)^p$
- Infinite-dimensional feature mappings are possible
  - E.g., RBF kernel: $k_{RBF}(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$
- Kernel functions for time series, strings, graphs, ...
- For a given kernel matrix, the Mercer map provides a feature mapping.
Summary

■ Representor Theorem: \( f_{\theta}^*(x) = \sum_{i=1}^{n} \alpha_i^* \phi(x_i)^T \phi(x) \)
  - Instances only interact through inner products
  - Great for few instances, many attributes

■ Kernel learning algorithms:
  - Kernel ridge regression
  - Kernel perceptron, SVM,
  - …