Some kernels for structured data

- Goal: construct a similarity score for objects such as
  - sequences
    - with variable length
    - by their interpretations
  - labeled graphs (or trees)
    - different size
    - different structure
  - other objects
    - by their interpretations

- The similarity must be a PD kernel
Rational kernels

- Compare sequences $x, y \in \Sigma^*$
  - $x = (0, 1, 1, 0, 1, 1, 0, 0)$
  - $y = (1, 1, 0, 1)$

- **Transducer**
  - maps a seq. $x$ sto seq. $z$ with a weight
  - defines a “weighed relation” $T(x, z) \rightarrow R$
  - is implemented by a *finite state automaton*

- Kernel
  - $x, y$ are similar if they are transduced often to the same $z$
    - $K(x,y) = \sum_z T(x, z) \cdot T(y, z)$

- Advantage
  - Given an automaton for $T$, can construct an automaton for $K$
Rational kernels: Implementation

- Automaton for $K(x, y)$
  - invert $T$
  - compose $T$ and $T^{-1}$
Rational kernels: Examples

- **Bag-of-subsequences**
  - \( x \) binary sequence
  - \( z \) binary sequence of 4 characters
  - \( T(x, z) = \# \) occurrences of \( z \) in \( x \)
  - \( K(x, y) = \sum_z T(x, z) T(y, z) \) is large iff \( x, y \) contain similar subsequences

- **Normalization**
  \[
  K(x, y) / (K(x, x) K(y, y))^{1/2}
  \]

- **Other examples**
  - HMM-like models

\[
\begin{align*}
x & = (0, 1, 1, 0, 1, 1, 0) \\
T & \quad \quad z \\
2 & \quad 0, 1, 1, 0 \\
1 & \quad 1, 1, 0, 1 \\
1 & \quad 1, 0, 1, 1 \\
y & = (1, 1, 0, 1) \\
T & \quad \quad z \\
1 & \quad 1, 1, 0, 1 \\
K(x, y) & = 1
\end{align*}
\]
Convolution kernels

- To compare objects $x, y$
  - decompose each object in $d$ components
  - compare components and combine results
  - (repeat recursively until atomic components)

- Example: tree

$x$ is a $d$-degree tree

Subpart relation

$$R(x_1, x_2, \ldots, x_d, x)$$

$$K(x, y) = \prod_{i=1}^{d} K_i(x_i, y_i)$$
Convolution kernels

- Example: string
  - \( \mathbf{x} \) is a string
  - Subpart relation
    \( R(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}) \) iif
      \( \mathbf{x}_1, \mathbf{x}_2 \) are (non-empty) strings such that \( \mathbf{x} = \text{concat}(\mathbf{x}_1, \mathbf{x}_2) \)

- Multiple decompositions are possible
  - \( R^{-1}(\mathbf{x}) = \{ (\mathbf{x}_1, \mathbf{x}_2) : R(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}) \} \)

- Convolution kernel

\[
k(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{x}' \in R^{-1}(\mathbf{x})} \sum_{\mathbf{y}' \in R^{-1}(\mathbf{y})} \prod_{i=1}^{r} k_i(\mathbf{x}'_i, \mathbf{y}'_i)
\]
We can represent the relation ”$x_1, \ldots, x_d$ are the parts of $x$” by a relation $R$ on the set $X_1 \times \cdots \times X_D \times X$, where $R(x_1, \ldots, x_D, x)$ is true iff $x_1, \ldots, x_D$ are the parts of $x$. For brevity, let $\vec{x} = x_1, \ldots, x_D$, and denote $R(x_1, \ldots, x_D, x)$ by $R(\vec{x}, x)$. Let $R^{-1}(x) = \{\vec{x} : R(\vec{x}, x)\}$. We say $R$ is finite if $R^{-1}(x)$ is finite for all $x \in X$. Here are some examples:

1. If $x$ is a $D$-tuple in $X = X_1 \times \cdots \times X_D$, and each component of $x \in X$ is a part of $x$, then $R(\vec{x}, x)$ iff $\vec{x} = x$.

2. If $X_1 = X_2 = X$, where $X$ is the set of all finite strings over a finite alphabet $\mathcal{A}$, then we can define $R(x_1, x_2, x)$ iff $x_1 \circ x_2 = x$, where $x_1 \circ x_2$ denotes the concatenation of strings $x_1$ and $x_2$.

3. Continuing the previous example, if the alphabet $\mathcal{A}$ has only one letter, then a finite string can be represented by the nonnegative integer $n$ that is its length, so $X_1 = X_2 = X = \{0, 1, \ldots\}$ and $R(n_1, n_2, n)$ iff $n_1 + n_2 = n$.

4. If $X_1 = \ldots = X_D = X$, where $X$ is the set of all $D$-degree ordered and rooted trees, then we can define $R(\vec{x}, x)$ iff $x_1, \ldots, x_D$ are the $D$ subtrees of the root of the tree $x \in X$. 
Kernels based on local info

- **Given**
  - \{ x_1, ..., x_n \} collection of objects
  - “local” distances
    formally: \( G \) undirected weighed DAG

- **Get geodesic distances** \( D \)
  - all shortest-paths \( D \)
  - regularize by finding low-dimensional embedding
    (ISOMAP)

- **Get a kernel**
  - Use identity
    \( D(x_1,x_2) = K(x_1,x_1) + K(x_2,x_2) - 2K(x_1,x_2) \)
  - Make *positive definite* by incrementing the diagonal
    \( K \leftarrow K + \lambda I \)

See references in
Graph kernels

- Compare *labeled graphs* \( x, y \in \Sigma^* \)
  - given a kernel on *paths* \( k_{\text{path}}(h, h') \)
  - extend to kernel on graphs
  - try to capture “topology”

- Compare all paths \( W(G_1), W(G_2) \)

\[
k_G(G_1, G_2) = \sum_{h \in W(G_1)} \sum_{h' \in W(G_2)} k_{\text{path}}(h, h')
\]

- walks (any path)
- proper paths (no self intersection)
- shortest paths
- random walks

Fisher kernels

• Compare objects $x, y$ by a **generative model**
  - given $p(x \mid \theta)$
  - map points $x$ to maximum-likelihood parameters $\theta_x$
  - compare $K(\theta_x, \theta_y)$

• Local analysis
  - log-likelihood function $L(x, \theta) = \log p(x \mid \theta)$
  - assume $x \sim p(x \mid \theta)$
  - maximum likelihood is consistent $\forall \hat{\theta}: E[L(x, \hat{\theta})] \leq E[L(x, \theta)]$

• Fisher score
  
  $$U(x, \theta) = \nabla_\theta L(x, \theta) \quad E[U(x, \theta)] = \frac{\partial}{\partial \theta} E[L(x, \theta)] = 0$$

• Fisher information
  
  $$I(\theta) = E[U(x, \theta)^2] = \text{var} U(x, \theta)$$
Fisher kernels

- Fisher information matrix as approx. second derivative
  \[
  E \left[ \frac{\partial^2}{\partial \theta^2} L(x, \theta) \right] = E \left[ \frac{1}{p(x|\theta)} \frac{\partial^2}{\partial \theta^2} p(x|\theta) \right] - E \left[ \left( \frac{1}{p(x|\theta)} \frac{\partial}{\partial \theta} p(x|\theta) \right)^2 \right] 
  \approx -E \left[ \left( \frac{\partial}{\partial \theta} \log p(x|\theta) \right)^2 \right] 
  = -E[U(x, \theta)^2] = -I(\theta)
  \]

- Approx. ML estimate
  \[
  L(x, \theta + \delta \theta) \approx L(x, \theta) + U(x, \theta) \delta \theta - \frac{1}{2} I(\theta)(\delta \theta)^2 
  \]
  \[
  \delta \theta_x \approx I(\theta)^{-1} U(x, \theta)
  \]

- Fisher kernel
  \[
  K(x, y) = \delta \theta_x I(\theta) \delta \theta_y = U(x, \theta)I(\theta)^{-1} U(y, \theta)
  \]
Invariance

• Why weighting by \( I \)?

\[
K(x, y) = \delta \theta_x I(\theta) \delta \theta_y = U(x, \theta) I(\theta)^{-1} U(y, \theta)
\]

• Reparametrization \( \theta = \phi(\lambda) \)

\[
L'(x, \lambda) = L(x, \phi(\lambda)) \quad U'(x, \lambda) = U(x, \phi(\lambda)) \dot{\phi}(\lambda)
\]

\[
I'(\lambda) = \dot{\phi}(\lambda) I(\phi(\lambda)) \dot{\phi}(\lambda)
\]

• Fisher kernel is invariant to reparametrization

\[
K(x, y) = U'(I')^{-1} U' = U \phi \phi^{-1} I \phi^{-1} \phi U = UI^{-1} U
\]
Tutorial

- MediaLandscape Player