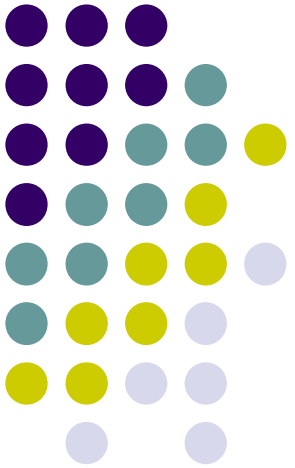
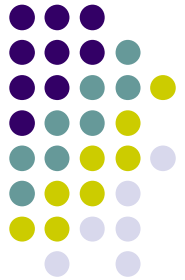
Protein function prediction via graph kernels

Jing Tang

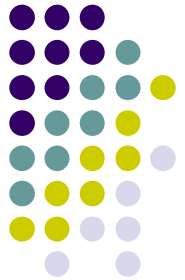


Overview



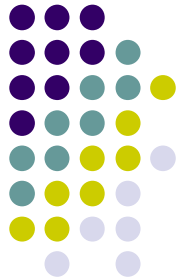
- Review on Borgwardt *et al*., 2005. Bioinformatics.
- Prediction of protein functions using sequence and structure information
- Graph kernel and hyperkernel techniques

Biological motivation



- Determination of a protein function according to its sequence and structure remains a daunting task.
- A protein is assumed to perform the same function as the most similar proteins in a database of known proteins.
- Question: How to define similarity? Sequence alignment, structure, common binding sites, chemical features …

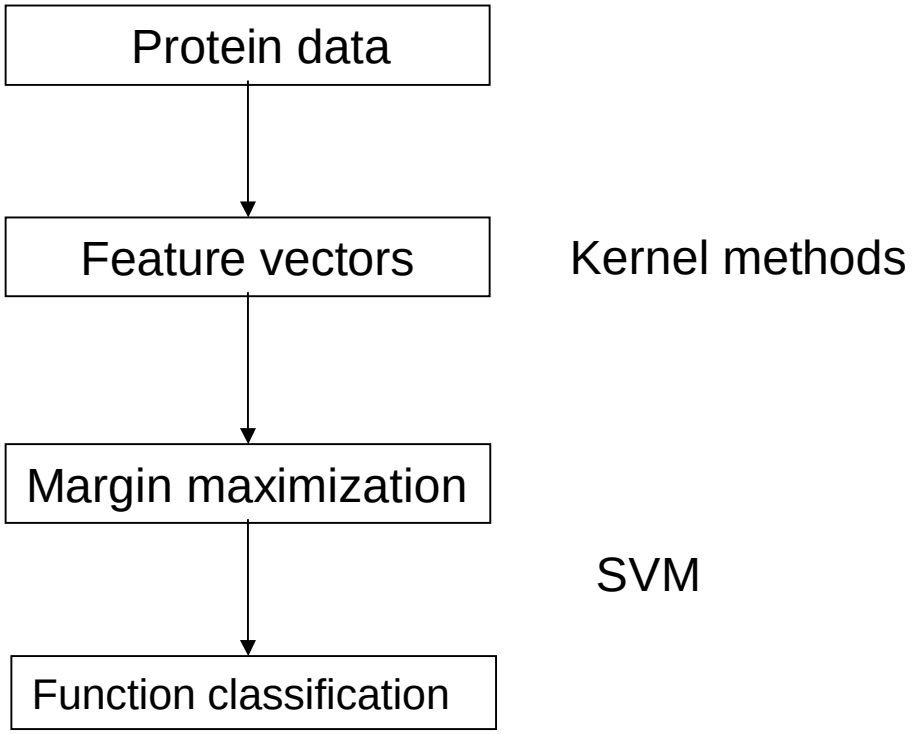
However, …



- There are no universally valid standards in defining similarity.
- Similarity in one aspect or another does not guarantee similar functions.
- Therefore, multiple similarity measures need to taken in combination.

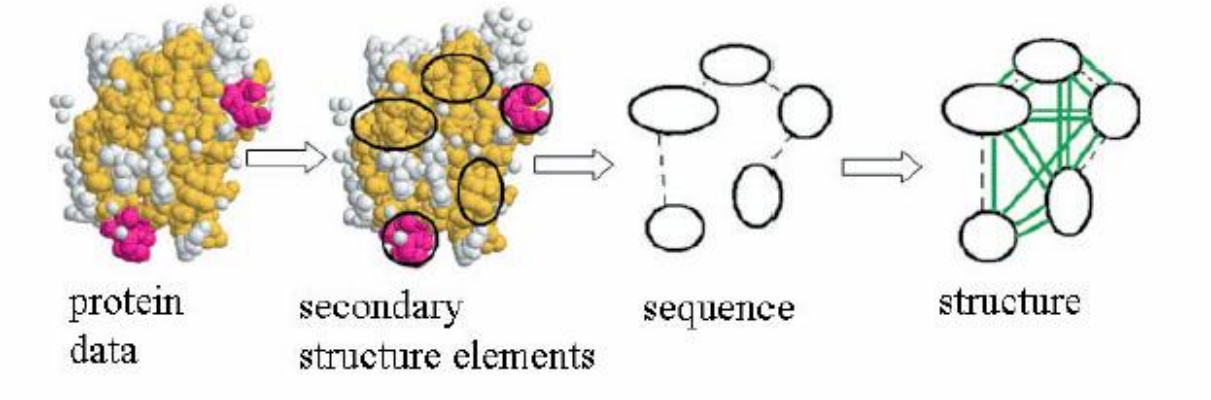
Solution

- Kernel methods and support vector machines

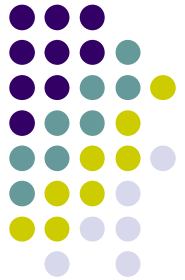


Why bother a new kernel?

- Existing kernel methods simply transform protein data into a simplified feature vector description, where detailed information was lost.
- Graph kernel provides a natural way to capture the protein structure information.

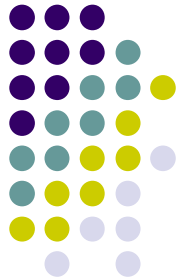


Graph terms



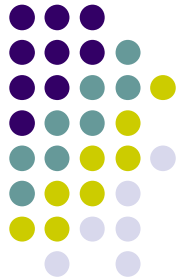
- Graph G: G(V,E)
- Attributed graph
- Adjacency matrix
- Walk in a graph

Protein graph representation



- Each graph represents one protein.
- Nodes represent Secondary structure elements (SSE).
- Edges represent either the actual linkage along the (amino acid) sequence, or the spatial neighbors in the structure
- Every node is connected to its three nearest spatial neighbors.(?)

Protein graph representation



- Sequential and structural information are represented as attributes.
- Node attributes
  - Type. (Helix, sheet, turn, …)
  - Length. (in amino acid sequence or in angstroms)
- Edge attributes
  - Type.
  - Length.

Random walk graph kernel

- Random walk counts the number of matching between two labeled graphs.
- The match is determined by comparing the attribute values along the walk path.

$$k_{\text{graph}}(G_1, G_2) = \sum_{\text{walk}_1 \in G_1} \sum_{\text{walk}_2 \in G_2} k_{\text{walk}}(\text{walk}_1, \text{walk}_2).$$

Direct product graph

- Designed for facilitating the computation of random walk kernel.
- Compared via a Dirac kernel (exact match).

$$V_{\times}(G_1 \times G_2) = \{(v_1, w_1) \in V \times W : (label(v_1) = label(w_1))\},$$

$$E_{\times}(G_1 \times G_2) = \{((v_1, w_1), (v_2, w_2)) \in V^2(G_1 \times G_2) : (v_1, v_2) \in E \wedge (w_1, w_2) \in F \wedge (label(v_1, v_2) = label(w_1, w_2))\}.$$

Computation of random walk kernel

- The adjacency matrix of the direct product graph can be used for computing random walk kernel

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{V_\times} \left[\sum_{n=0}^{\infty} \lambda^n A_\times^n \right]_{ij}.$$

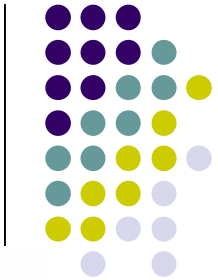
Modified random walk kernel

- The nodes in the protein graph contain continuous attributes such that an exact match required in the direct product graph is not applicable.
- Therefore, the adjacency matrix has to be continuous.
- This is done by introducing step kernel.

$$[A_\times]_{((v_i,w_i),(v_j,w_j))} = \begin{cases} k_{step}((v_i,v_j),(w_i,w_j)) \\ \quad if\ ((v_i,v_j),(w_i,w_j)) \in E_\times, \\ 0 \quad otherwise \end{cases}$$

$$with\ E_\times = E_\times(G_1 \times G_2)\ and\ (v_i,v_j) \in E\ and\ (w_i,w_j) \in F.$$

Step kernel



DEFINITION 4 (*Step kernel*). *For* $i \in \{1, \ldots, n-1\}$, *the step kernel is defined as*

$$k_{step}((v_i, v_{i+1}), (w_i, w_{i+1})) = k_{node}(v_i, w_i) * k_{node}(v_{i+1}, w_{i+1}) * k_{edge}((v_i, v_{i+1}), (w_i, w_{i+1})),$$

where k_{edge} *is defined as*

$$k_{edge}((v_i, v_{i+1}), (w_i, w_{i+1})) = k_{type}((v_i, v_{i+1}), (w_i, w_{i+1})) * k_{length}((v_i, v_{i+1}), (w_i, w_{i+1}))$$

and for $i \in \{1, \ldots, n\}$, k_{node} *is defined as*

$$k_{node}(v_i, w_i) = k_{type}(v_i, w_i) * k_{node\ labels}(v_i, w_i) * k_{length}(v_i, w_i).$$

Three component kernels

- Type kernel

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

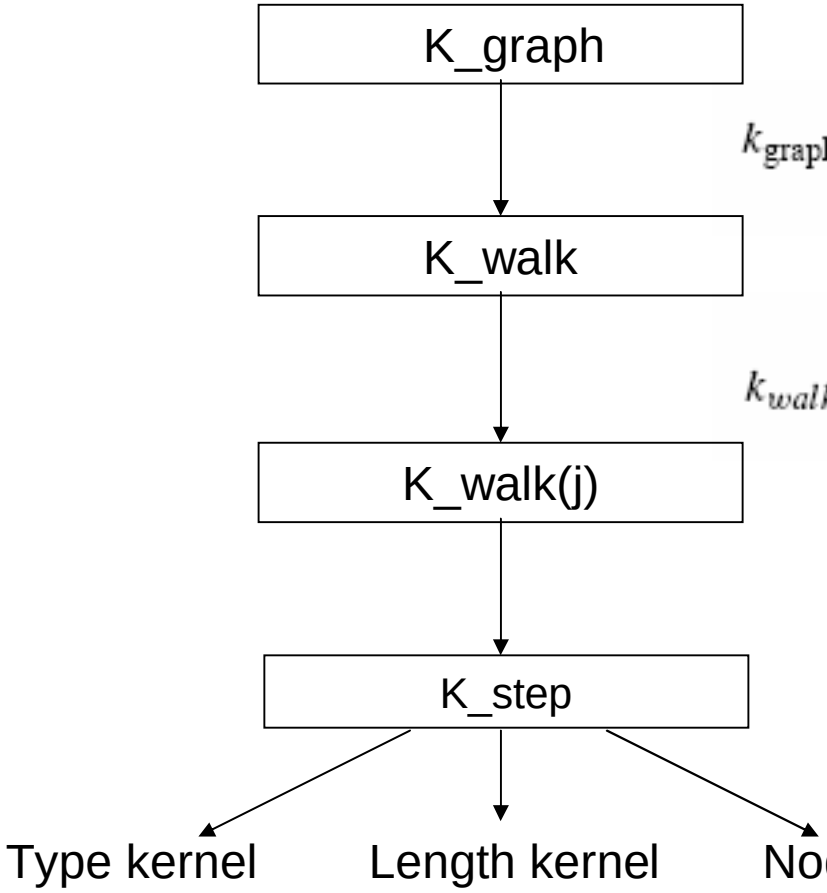
- Length kernel

$$k_{length}(x, x') = max(0, c - |length(x) - length(x')|).$$

- Node labels kernel

$$k_{node\ labels}(x, x') = \exp\left(-\frac{\|labels(x) - labels(x')\|^2}{2\sigma^2}\right).$$

Prove of positive definiteness

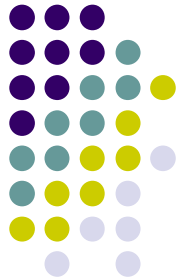


$$k_{\text{graph}}(G_1, G_2) = \sum_{\text{walk}_1 \in G_1} \sum_{\text{walk}_2 \in G_2} k_{\text{walk}}(\text{walk}_1, \text{walk}_2).$$

$$k_{walk}(walk_1, walk_2) = \prod_{i=1}^{n-1} k_{step}((v_i, v_{i+1}), (w_i, w_{i+1})).$$

$$\begin{aligned} & k_{step}((v_i, v_{i+1}), (w_i, w_{i+1})) \\ & \quad = k_{node}(v_i, w_i) * k_{node}(v_{i+1}, w_{i+1}) \\ & \quad * k_{edge}((v_i, v_{i+1}), (w_i, w_{i+1})), \end{aligned}$$

Hyperkernels



- A trick to choose best kernel (informative attributes ?)
- Offset by controlling the kernel complexity
- Obtained through minimizing a regularized quality functional (?)
- Semidefinte programming (SDP) (?) implemented in MATLAB/SVLAB

Discussions