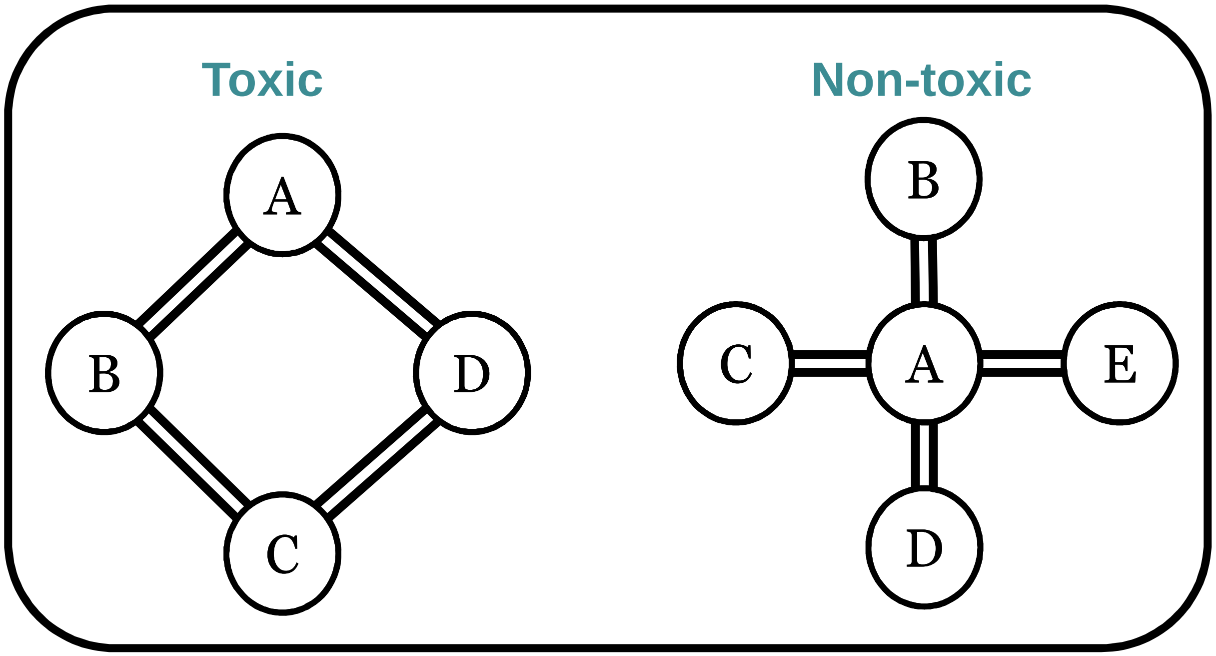
Graph Classification

Classification Outline

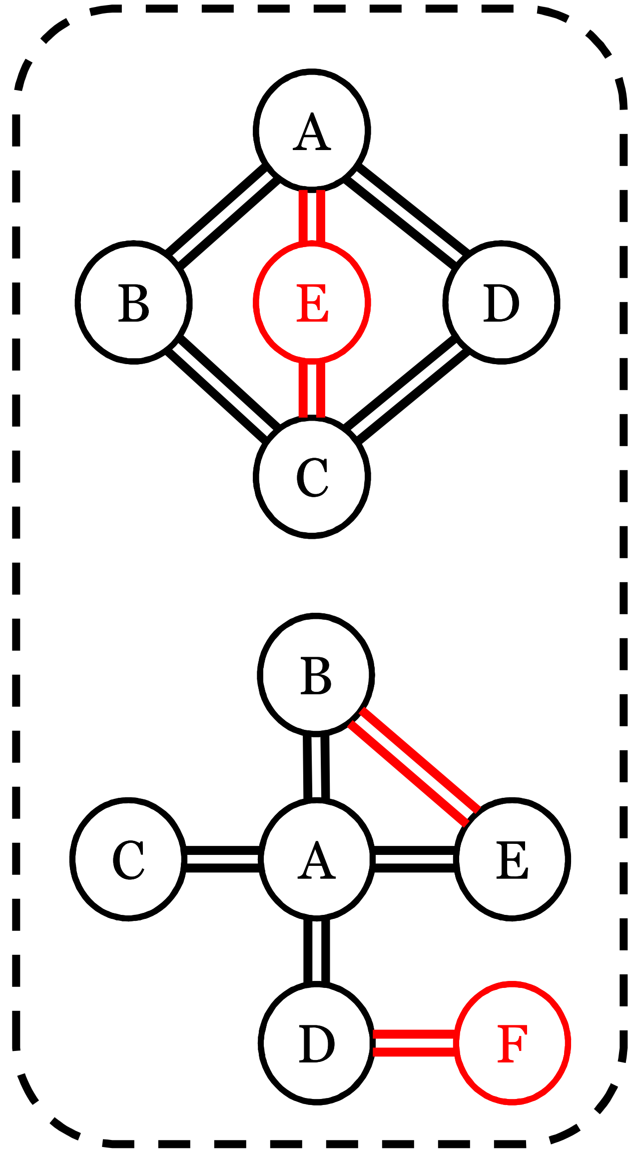
- **Introduction, Overview**
- **Classification using Graphs**
  - Graph classification – Direct Product Kernel
    - Predictive Toxicology example dataset
  - Vertex classification – Laplacian Kernel
    - WEBKB example dataset
- **Related Works**

Example: Molecular Structures

Known



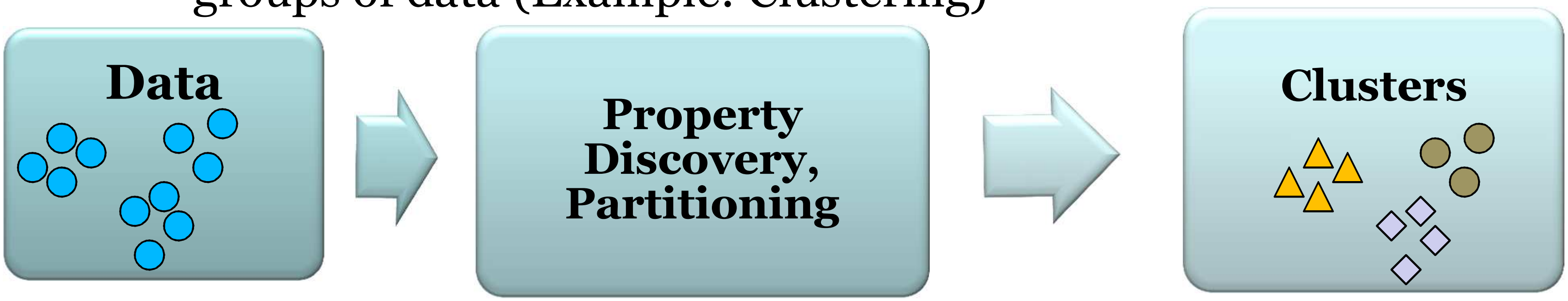
Unknown



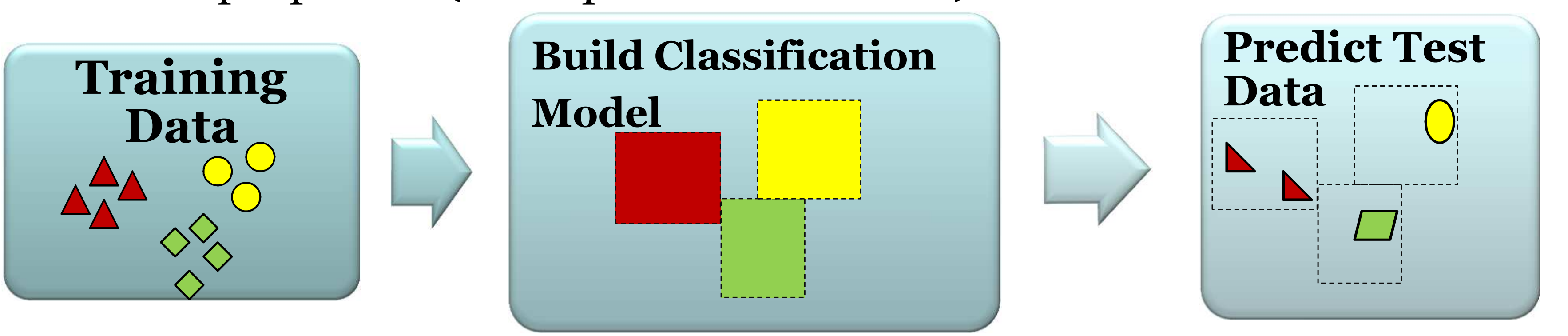
Task: predict whether molecules are toxic, given set of known examples

Solution: Machine Learning

- **Computationally *discover* and/or *predict* properties of interest of a set of data**
- **Two Flavors:**
  - **Unsupervised**: discover discriminating properties among groups of data (Example: Clustering)

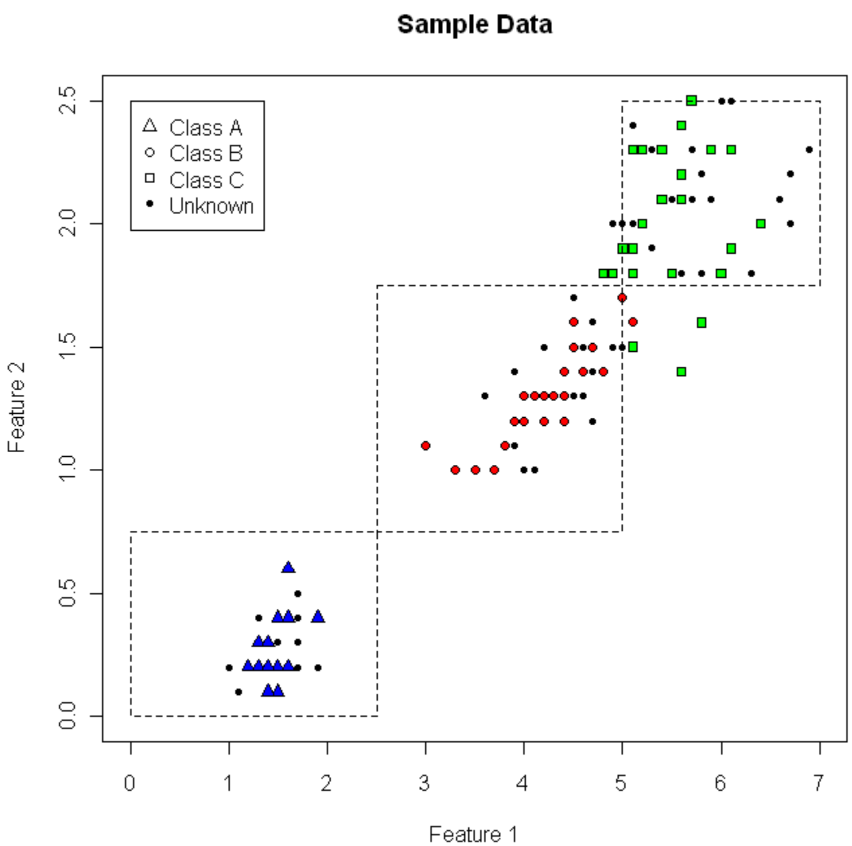


  - **Supervised**: known properties, categorize data with unknown properties (Example: Classification)

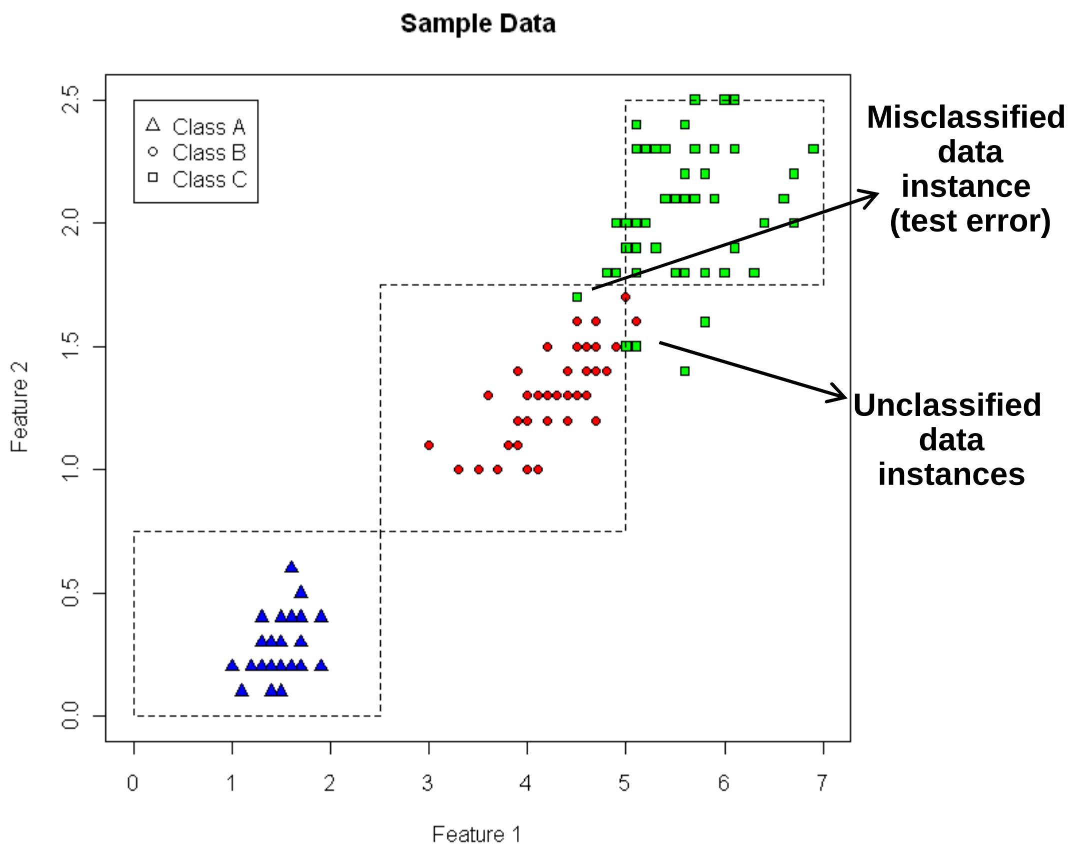


Classification

- ***Classification*: The task of assigning class labels in a discrete class label set Y to input instances in an input space X**
- **Ex: Y = { *toxic, non-toxic* }, X = {*valid molecular structures*}**



Training the classification model using the training data



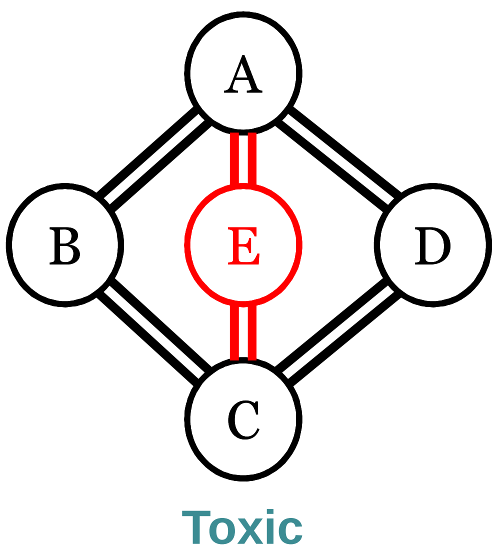
Assignment of the unknown (test) data to appropriate class labels using the model

Classification Outline

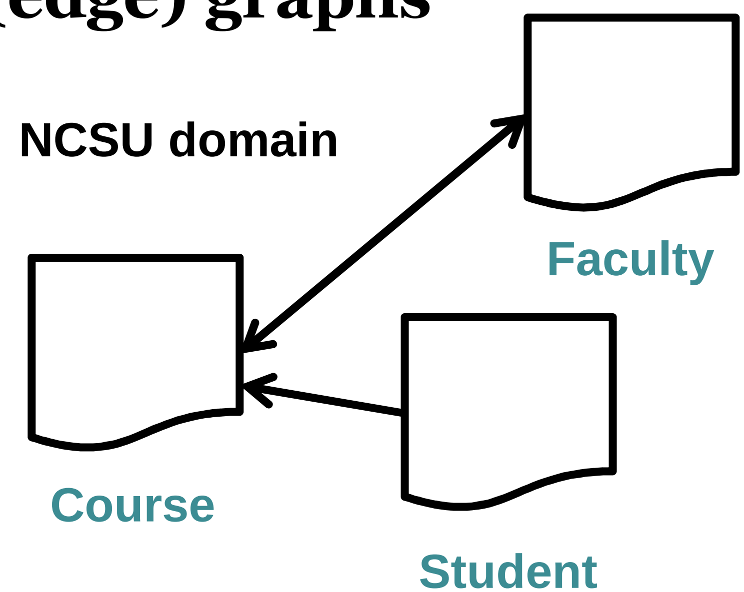
- **Introduction, Overview**
- **Classification using Graphs,**
  - Graph classification – Direct Product Kernel
    - Predictive Toxicology example dataset
  - Vertex classification – Laplacian Kernel
    - WEBKB example dataset
- **Related Works**

Classification with Graph Structures

- **Graph classification (between-graph)**
  - Each full graph is assigned a class label
- **Example: Molecular graphs**



- **Vertex classification (within-graph)**
  - Within a single graph, each vertex is assigned a class label
- **Example: Webpage (vertex) / hyperlink (edge) graphs**



Relating Graph Structures to Classes?

- **Frequent Subgraph Mining (Chapter 7)**
  - Associate frequently occurring subgraphs with classes
- **Anomaly Detection (Chapter 11)**
  - Associate anomalous graph features with classes
- ***Kernel-based methods (Chapter 4)**
  - Devise kernel function capturing graph similarity, use vector-based classification via the *kernel trick*

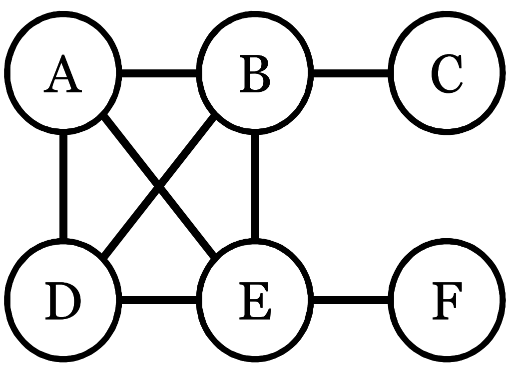
Relating Graph Structures to Classes?

- **This chapter focuses on kernel-based classification.**
- **Two step process:**
  - Devise kernel that captures property of interest
  - Apply *kernelized* classification algorithm, using the kernel function.
- **Two type of graph classification looked at**
  - Classification of Graphs
    - Direct Product Kernel
  - Classification of Vertices
    - Laplacian Kernel
- **See Supplemental slides for *support vector machines* (SVM), one of the more well-known kernelized classification techniques.**

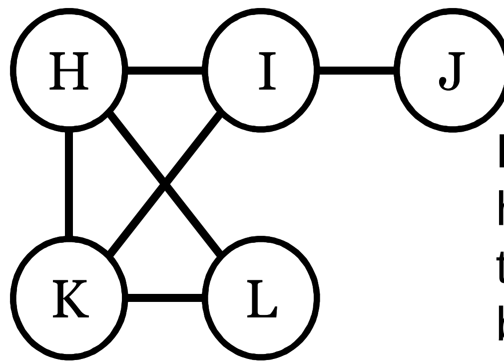
Walk-based similarity (Kernels Chapter)

- **Intuition – two graphs are similar if they exhibit similar patterns when performing random walks**

Random walk vertices heavily distributed towards A,B,D,E

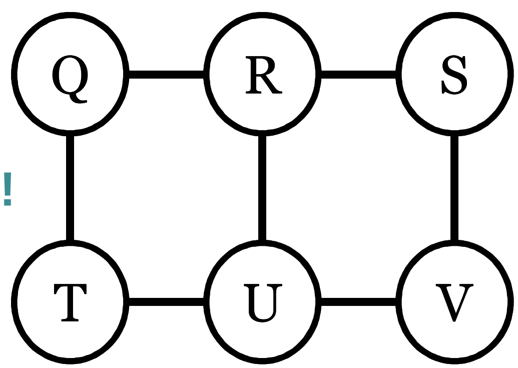


Similar!



Random walk vertices heavily distributed towards H,I,K with slight bias towards L

Not Similar!



Random walk vertices evenly distributed

Classification Outline

- **Introduction, Overview**
- **Classification using Graphs**
  - Graph classification – Direct Product Kernel
    - Predictive Toxicology example dataset.
  - Vertex classification – Laplacian Kernel
    - WEBKB example dataset.
- **Related Works**

Direct Product Graph – Formal Definition

Input Graphs

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

Direct Product Notation

$$G_X = G_1 \times G_2$$

Intuition

Vertex set: each vertex of V_1 paired with *every* vertex of V_2

Edge set: Edges exist only if both pairs of vertices in the respective graphs contain an edge

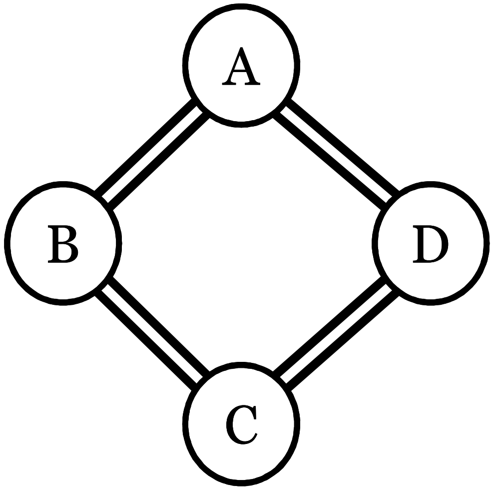
Direct Product Vertices

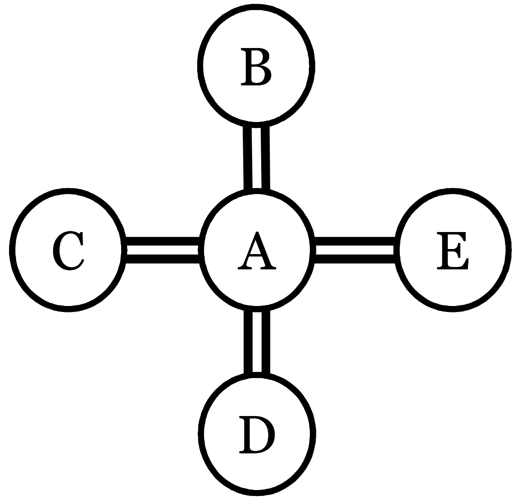
$$V(G_x) = \{(a, b) \in V_1 \times V_2\}$$

Direct Product Edges

$$E(G_x) = \{\big((a, b), (c, d)\big) | (a, c) \in E_1 and\ (b, d) \in E_2\}$$

Direct Product Graph - example





Type-A **Type-B**

Type-A	A	B	C	D
A	0	1	1	0
B	1	0	0	1
C	1	0	0	1
D	0	1	1	0

Type-B	A	B	C	D	E
A	0	1	1	1	1
B	1	0	0	0	0
C	1	0	0	0	0
D	1	0	0	0	0
E	1	0	0	0	0

Direct Product Graph Example

Type-A	A	B	C	D
A	0	1	1	0
B	1	0	0	1
C	1	0	0	1
D	0	1	1	0

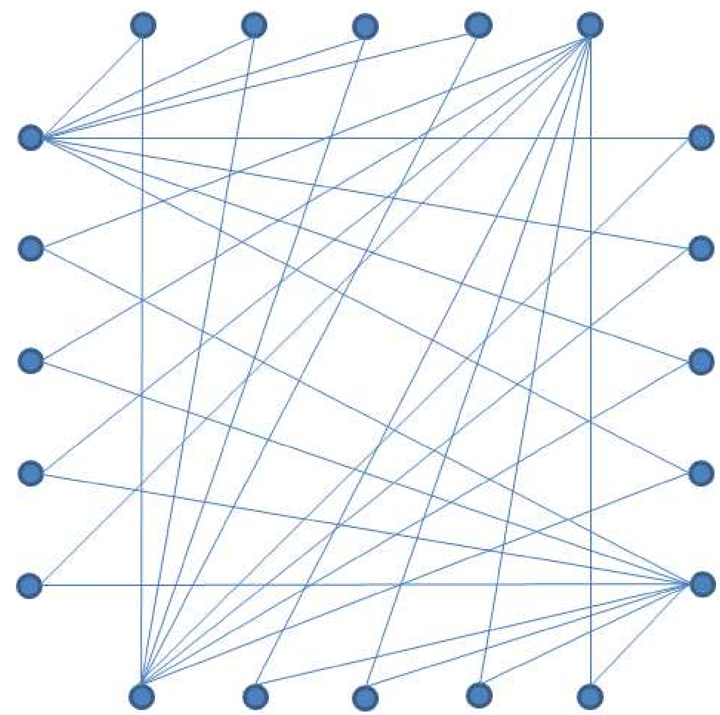
Type-B	A	B	C	D	E
A	0	1	1	1	1
B	1	0	0	0	0
C	1	0	0	0	0
D	1	0	0	0	0
E	1	0	0	0	0

Intuition: multiply each entry of Type-A by ***entire matrix*** of Type-B

Type-A		A					B					C					D				
	Type-B	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
A	A	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	0	0
	B	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	D	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
B	A	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	B	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	C	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	D	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
C	A	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	B	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	C	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	D	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
D	A	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	0	0
	B	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	D	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Direct Product Kernel (see Kernel Chapter)

1. **Compute direct product graph G_x**
2. **Compute the maximum in- and out-degrees of *Gx*, *di* and *do*.**
3. **Compute the decay constant γ < 1 / min(*di*, *do*)**
4. **Compute the infinite weighted geometric series of walks (array *A*).**
5. **Sum over all vertex pairs.**



Direct Product Graph of Type-A and Type-B

$$k(G_1, G_2) = \sum_{i,j} (I - \frac{A_{ij}}{\gamma})^{-1}$$

Kernel Matrix

$$\begin{bmatrix} K(G_1, G_1), K(G_1, G_2), \dots, K(G_1, G_n) \\ K(G_2, G_1), K(G_2, G_2), \dots, K(G_2, G_n) \\ \dots \\ K(G_n, G_1), K(G_n, G_2), \dots, K(G_n, G_n) \end{bmatrix}$$

- **Compute direct product kernel for all pairs of graphs in the set of known examples.**
- **This matrix is used as input to SVM function to create the classification model.**
  - ***** Or any other kernelized data mining method!!!**

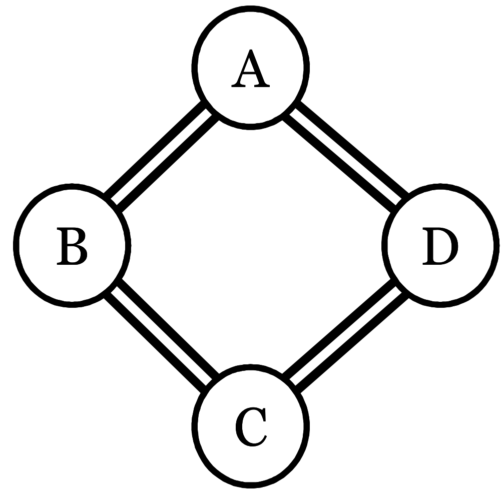
Classification Outline

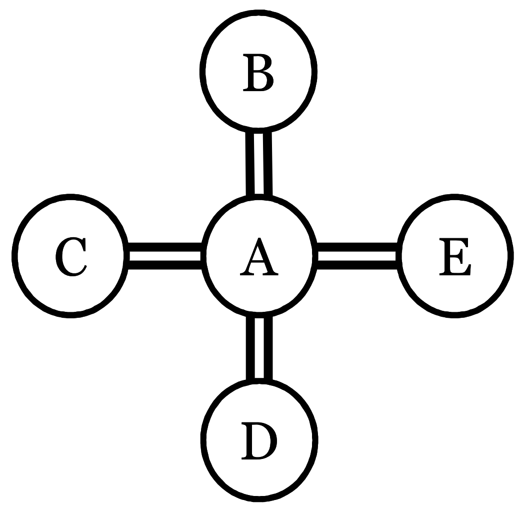
- **Introduction, Overview**
- **Classification using Graphs,**
  - Graph classification – Direct Product Kernel
    - Predictive Toxicology example dataset.
  - Vertex classification – Laplacian Kernel
    - WEBKB example dataset.
- **Related Works**

Predictive Toxicology (PTC) dataset

- The PTC dataset is a collection of molecules that have been tested positive or negative for toxicity.

```
# R code to create the SVM model
data("PTCData") # graph data
data("PTCLabels") # toxicity information
# select 5 molecules to build model on
sTrain = sample(1:length(PTCData),5)
PTCDataSmall <- PTCData[sTrain]
PTCLabelsSmall <- PTCLabels[sTrain]
# generate kernel matrix
K = generateKernelMatrix (PTCDataSmall,
PTCDataSmall)
# create SVM model
model =ksvm(K, PTCLabelsSmall,
kernel='matrix')
```





Classification Outline

- **Introduction, Overview**
- **Classification using Graphs,**
  - Graph classification – Direct Product Kernel
    - Predictive Toxicology example dataset.
  - Vertex classification – Laplacian Kernel
    - WEBKB example dataset.
- **Related Works**

Kernels for Vertex Classification

- **von Neumann kernel**
  - **(Chapter 6)**

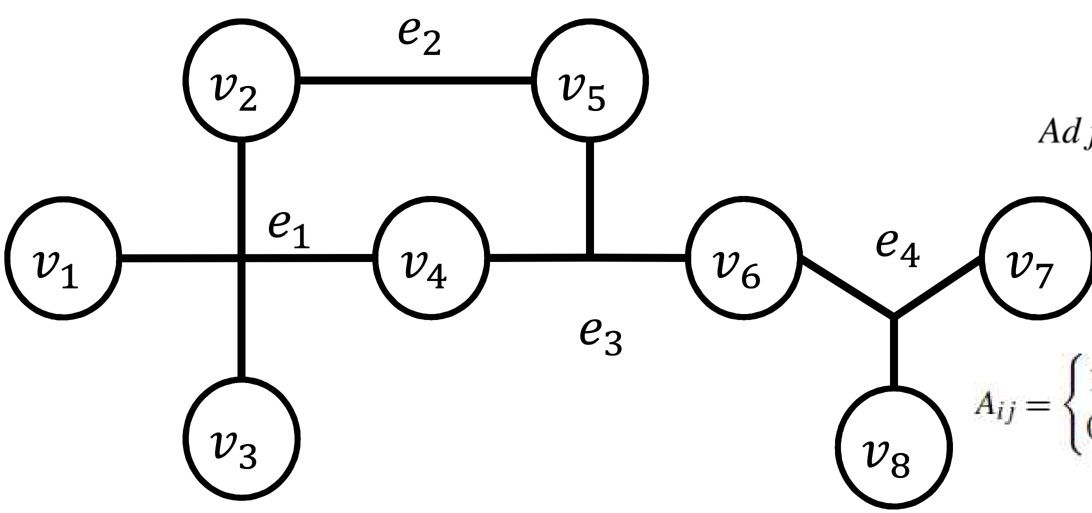
$$K = \sum_{i=1}^{\infty} \gamma^{i-1} (B^T B)^i$$

- **Regularized Laplacian**
  - **(This chapter)**

$$K = \sum_{i=1}^{\infty} \gamma^{i} (-L)^i$$

Example: Hypergraphs

- **A hypergraph is a generalization of a graph, where an edge can connect any number of vertices**
- **I.e., each edge is a *subset* of the vertex set.**



- **Example: word-webpage graph**
  - **Vertex – webpage**
  - **Edge – set of pages containing same word**

$$Adjacency\ Matrix\ A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{ij} = \begin{cases} 1, & \text{if and vertex } v_i \text{ belongs to edge } e_j \text{ in the hypergraph} \\ 0, & \text{otherwise.} \end{cases}$$

“Flattening” a Hypergraph

- **Given hypergraph matrix A, $A \times A^T$ represents “similarity matrix”**
- **Rows, columns represent vertices**
- **(i, j) entry – number of hyperedges incident on both vertex i and j.**
- **Problem: some neighborhood info. lost (vertex 1 and 3 just as “similar” as 1 and 2)**

$$Ad\,jacency\ Matrix\ A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$AA^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Laplacian Matrix

- **In the mathematical field of graph theory the Laplacian matrix (L), is a matrix representation of a graph.**
- **L = D – M**
  - **M – adjacency matrix of graph (e.g., $A*A^T$ from hypergraph flattening)**
  - **D – degree matrix (diagonal matrix where each (i,i) entry is vertex i's [weighted] degree)**
- **Laplacian used in many contexts (e.g., spectral graph theory)**

$$AA^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$D = \textstyle\sum_j [AA^T]_{ij}$$

$$D = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$L = D - AA^T$$

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 4 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Normalized Laplacian Matrix

- ***Normalizing* the matrix helps eliminate bias in matrix toward high-degree vertices**

$$L_{i,j} := \begin{cases} 1 & \text{if } i = j \text{ and } \deg(v_i) \neq 0 \\ \dfrac{-1}{\sqrt{\deg(v_i)\deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Original L

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 4 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Regularized L

$$L = \begin{bmatrix} 1.0 & -0.20 & -0.3 & -0.2 & 0.0 & 0.0 & 0.0 \\ -0.2 & 1.0 & -0.2 & -0.2 & -0.2 & 0.0 & 0.0 \\ -0.3 & -0.2 & 1.0 & -0.2 & 0.0 & 0.0 & 0.0 \\ -0.2 & -0.2 & -0.2 & 1.0 & -0.2 & 0.0 & 0.0 \\ 0.0 & -0.2 & 0.0 & -0.2 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -0.5 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.5 & 1.0 \end{bmatrix}$$

Laplacian Kernel

- **Uses walk-based geometric series, only applied to regularized Laplacian matrix**
- **Decay constant NOT degree-based – instead tunable parameter < 1**

$$K = \sum_{i=1}^{\infty} \gamma^i (-L)^i$$

$$K = (I + \gamma L)^{-1}$$

$$L = \begin{bmatrix} 1.0 & -0.20 & -0.3 & -0.2 & 0.0 & 0.0 & 0.0 \\ -0.2 & 1.0 & -0.2 & -0.2 & -0.2 & 0.0 & 0.0 \\ -0.3 & -0.2 & 1.0 & -0.2 & 0.0 & 0.0 & 0.0 \\ -0.2 & -0.2 & -0.2 & 1.0 & -0.2 & 0.0 & 0.0 \\ 0.0 & -0.2 & 0.0 & -0.2 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -0.5 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.5 & 1.0 \end{bmatrix}$$

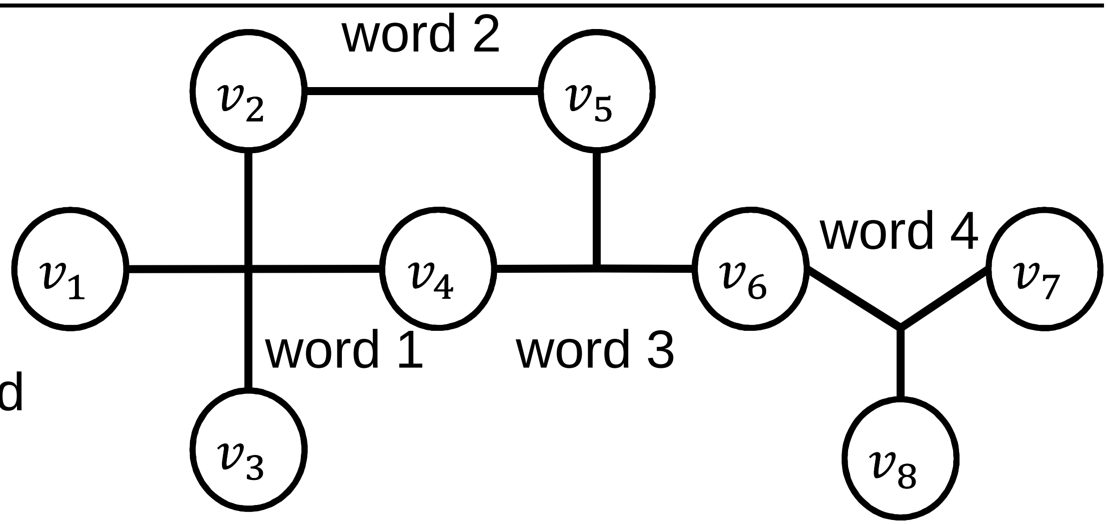
Regularized L

Classification Outline

- **Introduction, Overview**
- **Classification using Graphs,**
  - Graph classification – Direct Product Kernel
    - Predictive Toxicology example dataset.
  - Vertex classification – Laplacian Kernel
    - WEBKB example dataset.
- **Related Works**

WEBKB dataset

- The WEBKB dataset is a collection of web pages that include samples from four universities website.
- The web pages are assigned into five distinct classes according to their contents namely course, faculty, student, project and staff.
- The web pages are searched for the most commonly used words. There are 1073 words that are encountered at least with a frequency of 10.




```
# R code to create the SVM model
data(WEBKB)
# generate kernel matrix
K = generateKernelMatrixWithinGraph(WEBKB)
# create sample set for testing
holdout <- sample (1:ncol(K), 20)
# create SVM model
model =ksvm(K[-holdout,-holdout], y,
kernel='matrix')
```

Classification Outline

- **Introduction, Overview**
- **Classification using Graphs,**
  - Graph classification – Direct Product Kernel
    - Predictive Toxicology example dataset.
  - Vertex classification – Laplacian Kernel
    - WEBKB example dataset.
- **Kernel-based vector classification – Support Vector Machines**
- **Related Works**

Related Work – Classification on Graphs

- **Graph mining chapters:**
  - Frequent Subgraph Mining (Ch. 7)
  - Anomaly Detection (Ch. 11)
  - Kernel chapter (Ch. 4) – discusses in detail alternatives to the direct product and other “walk-based” kernels.
- **gBoost – extension of “boosting” for graphs**
  - Progressively collects “informative” frequent patterns to use as features for classification / regression.
  - Also considered a frequent subgraph mining technique (similar to gSpan in Frequent Subgraph Chapter).
- **Tree kernels – similarity of graphs that are trees.**

Related Work – Traditional Classification

- **Decision Trees**
  - Classification model → tree of conditionals on variables, where leaves represent class labels
  - Input space is typically a set of discrete variables
- **Bayesian belief networks**
  - Produces directed acyclic graph structure using Bayesian inference to generate edges.
  - Each vertex (a variable/class) associated with a probability table indicating likelihood of event or value occurring, given the value of the determined dependent variables.
- **Support Vector Machines**
  - Traditionally used in classification of real-valued vector data.
  - See Kernels chapter for kernel functions working on vectors.

Related Work – Ensemble Classification

- **Ensemble learning: algorithms that build multiple models to enhance stability and reduce selection bias.**
- **Some examples:**
  - Bagging: Generate multiple models using samples of input set (with replacement), evaluate by averaging / voting with the models.
  - Boosting: Generate multiple *weak* models, weight evaluation by some measure of model accuracy.

Related Work – Evaluating, Comparing Classifiers

- **This is the subject of Chapter 12, Performance Metrics**
- **A very brief, “typical” classification workflow:**
  1. Partition data into *training*, *test* sets.
  2. Build classification model using only the training set.
  3. Evaluate accuracy of model using only the test set.
- **Modifications to the basic workflow:**
  - Multiple rounds of training, testing (cross-validation)
  - Multiple classification models built (bagging, boosting)
  - More sophisticated sampling (all)

Related Work – Evaluating, Comparing Classifiers

- **This is the subject of Chapter 12, Performance Metrics**
- **A very brief, "typical" classification workflow:**
  1. Partition data into *training, test* sets.
  2. Build classification model using only the training set.
  3. Evaluate accuracy of model using only the test set.
- **Modifications to the basic workflow:**
  - Multiple rounds of training, testing (cross-validation)
  - Multiple classification models built (bagging, boosting)
  - More sophisticated sampling (all)