CPT-S 415

Big Data

Yinghui Wu

EME B45

CPT-S 415 Big Data

Data Mining & Graph Mining

- ✓ Classification
- ✓ Graph Classification
  - discrimitive feature-based classification
  - kernel-based classification

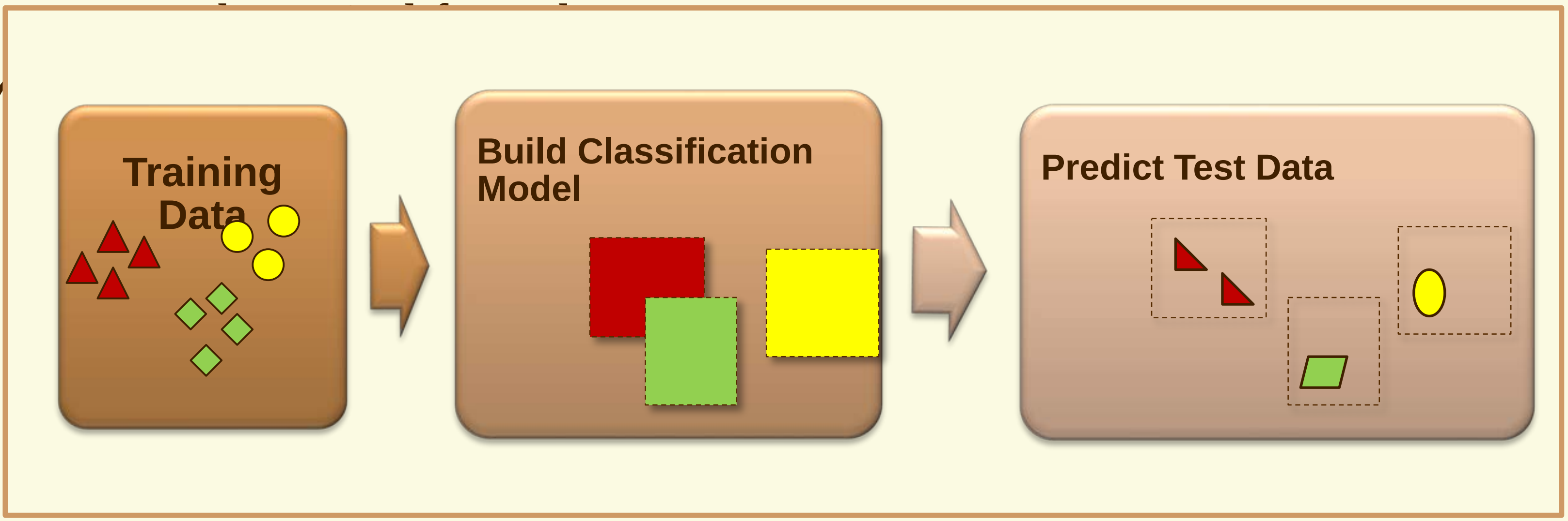
Classification

Classification vs prediction

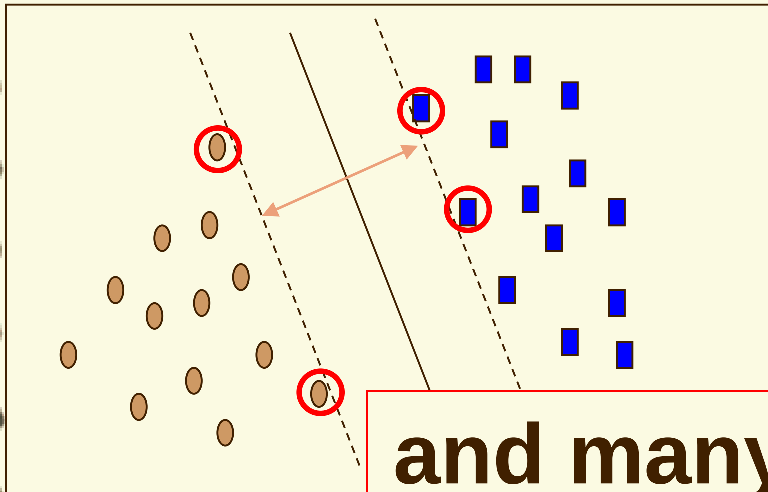
- ✓ Classification:
  - predicts categorical class labels
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- ✓ Prediction:
  - models continuous-valued functions, i.e., predicts unknown or missing values
- ✓ Typical Applications
  - credit approval
  - target marketing
  - medical diagnosis
  - treatment effectiveness analysis

Classification in two-steps

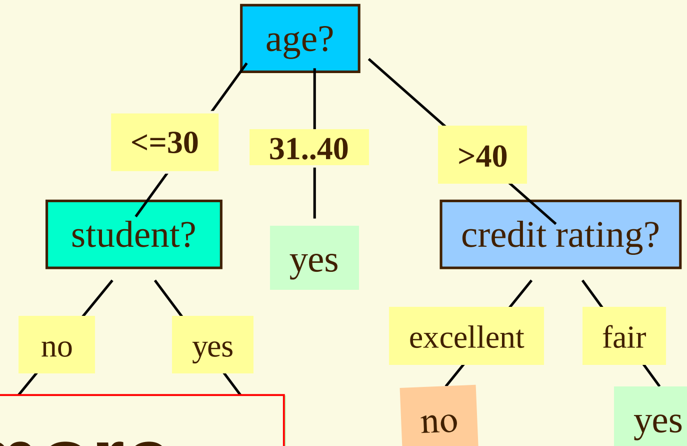
- ✓ Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction: training set
  - The model is represented as classification rules, decision trees, or



Existing Classification Methods

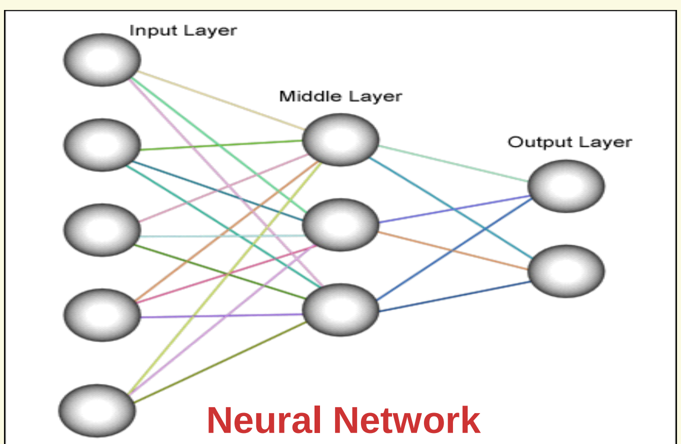


Support Vector Machine

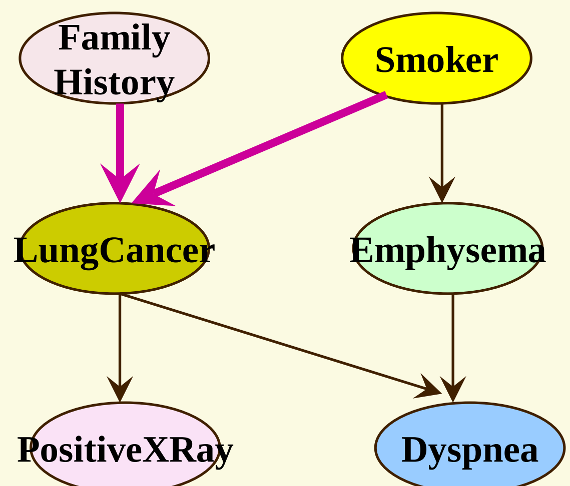


ecision Tree

and many more…



Neural Network



Bayesian Network

6

Classification by Decision Trees

- Decision tree
  - A flow-chart-like tree structure
  - Internal node denotes a test on an attribute
  - Branch represents an outcome of the test
  - Leaf nodes represent class labels or class distribution
- Decision tree generation consists of two phases
  - Tree construction
    - At start, all the training examples are at the root
    - Partition examples recursively based on selected attributes
  - Tree pruning
    - Identify and remove branches that reflect noise or outliers
- Use of decision tree: Classifying an unknown sample
  - Test the attribute values of the sample against the decision tree

Algorithm for Decision Tree Induction

- ✓ Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- ✓ Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Training Dataset

age	income	student	credit_rating
<=30	high	no	fair
<=30	high	no	excellent
31…40	high	no	fair
>40	medium	no	fair
>40	low	yes	fair
>40	low	yes	excellent
31…40	low	yes	excellent
<=30	medium	no	fair
<=30	low	yes	fair
>40	medium	yes	fair
<=30	medium	yes	excellent
31…40	medium	no	excellent
31…40	high	yes	fair
>40	medium	no	excellent

Extracting Classification Rules from Trees

- Represent the knowledge in the form of IF-THEN rules
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand
- Example

IF *age* = “<=30” AND *student* = “*no*” THEN *buys_computer* = “*no*”
IF *age* = “<=30” AND *student* = “*yes*” THEN *buys_computer* = “*yes*”
IF *age* = “31…40” THEN *buys_computer* = “*yes*”
IF *age* = “>40” AND *credit_rating* = “*excellent*” THEN *buys_computer* = “*yes*”
IF *age* = “>40” AND *credit_rating* = “*fair*” THEN *buys_computer* = “*no*”

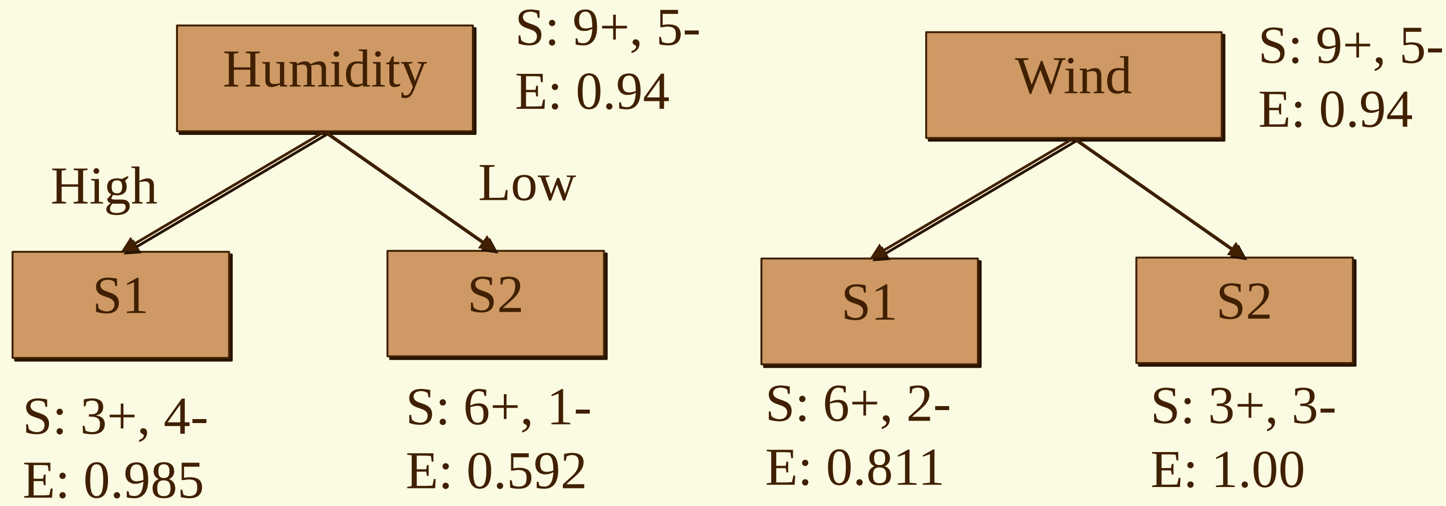
Over fitting?

Information Gain (ID3/C4.5)

- ✓ Select the attribute with the highest information gain
- ✓ Assume there are two classes, *P* and *N (e.g., yes/no)*
  - Let the set of examples *S* contain *p* elements of class *P* and *n* elements of class *N*
  - The amount of information, needed to decide if an arbitrary example in *S* belongs to *P* or *N* is defined as

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Information Gain in Decision Tree Induction



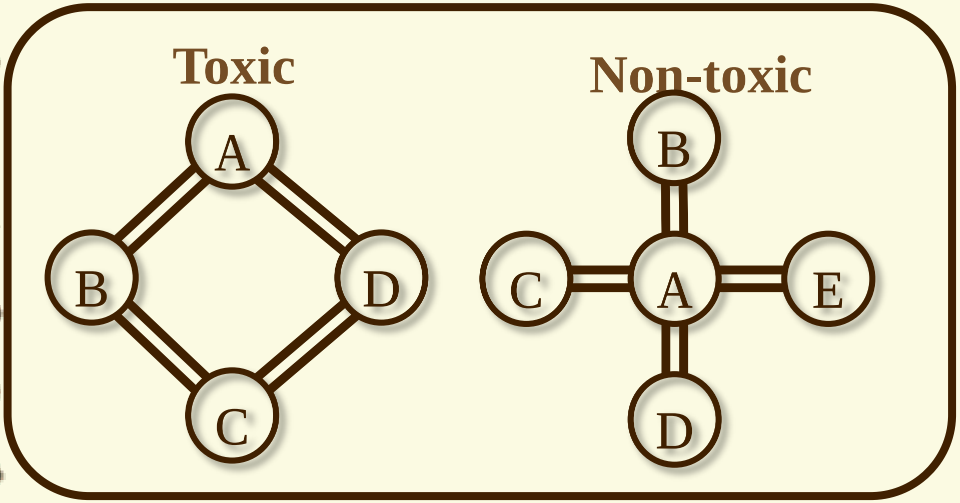
Gain (S, Humidity)=
.94- (7/14).985-(7/14).592=.151

Gain (S, Wind)=
.94- (8/14).811-(6/14).1.0=.048

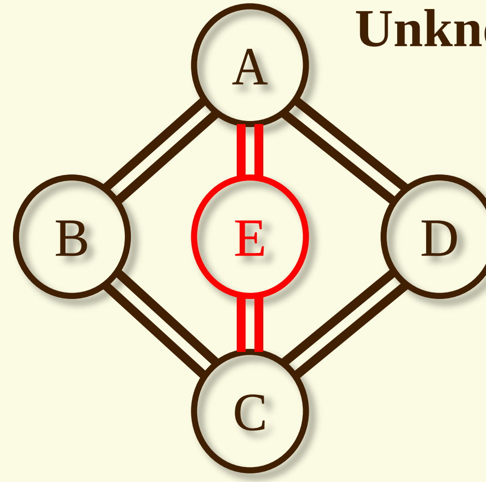
http://www.cs.princeton.edu/courses/archive/spr07/cos424/papers/mitchell-dectrees.pdf

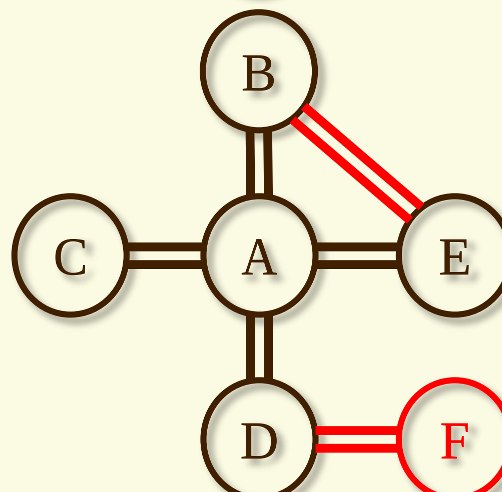
Molecular Structures classification

Known



Unknown

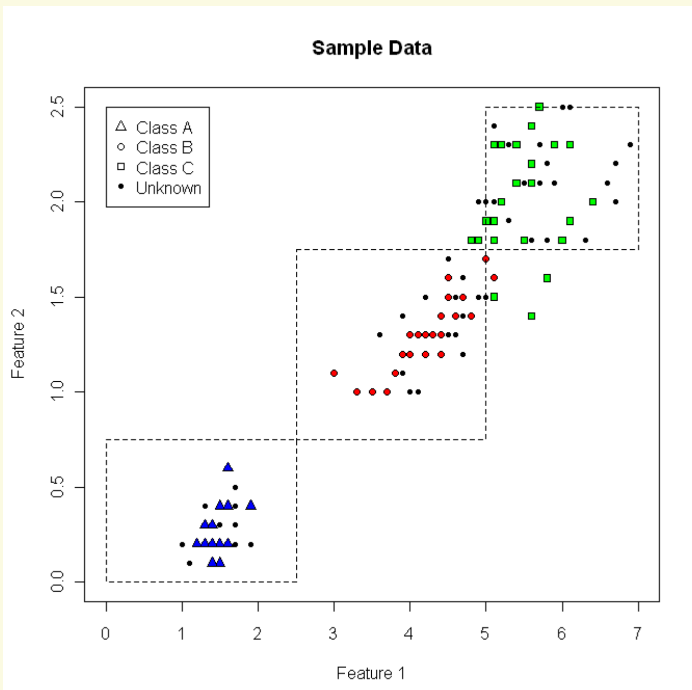




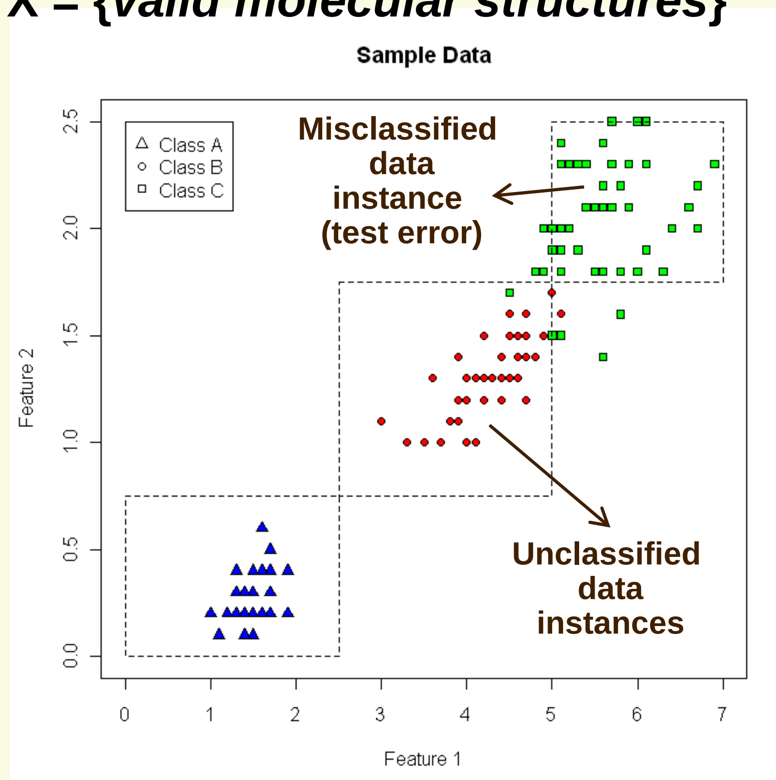
Task: predict whether molecules are toxic, given set of known examples

Classification

- **Task: 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



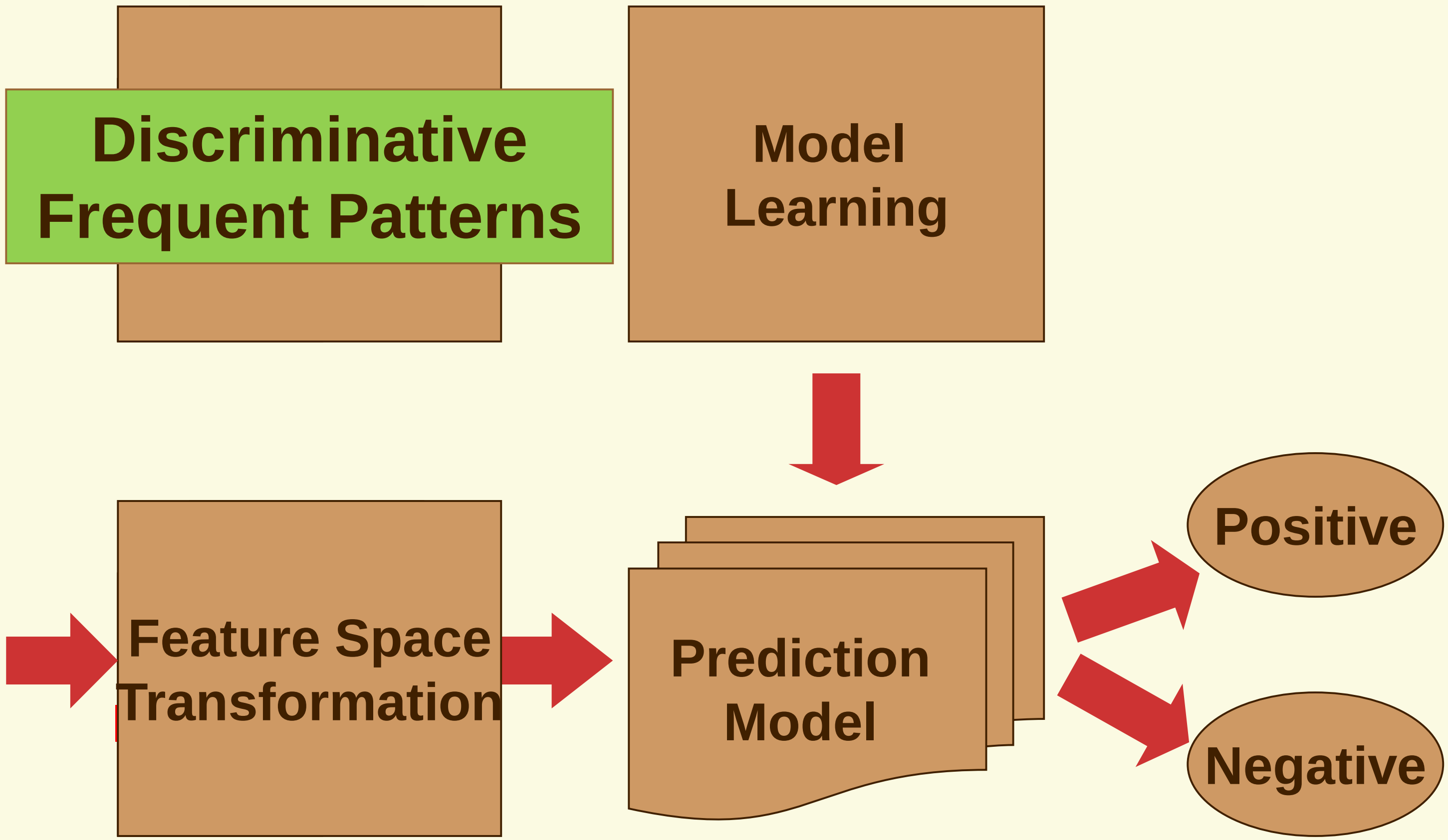
Assignment of the unknown (test) data to appropriate class labels using the model

Graph Classification Methods

- ✓ Graph classification – Structure Based/Frequent feature based
- ✓ Graph classification – Direct Product Kernel
  - Predictive Toxicology example dataset
- ✓ Vertex classification – Laplacian Kernel

Feature based Graph Classification

Discriminative Frequent Pattern-Based Classification



17

Pattern-Based Classification on Transactions

Attributes	Class
A, B, C	1
A	1
A, B, C	1
C	0
A, B	1
A, C	0
B, C	0

Mining

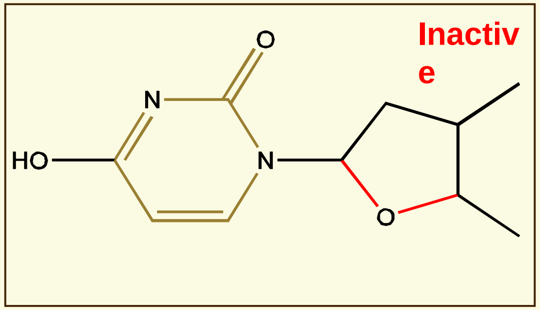
min_sup=3

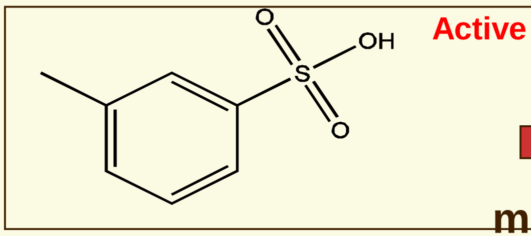
Frequent Itemset	Support
AB	3
AC	3
BC	3

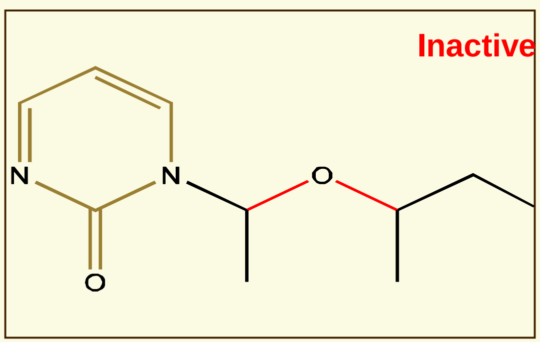
Augmented

A	B	C	AB	AC	BC	Class
1	1	1	1	1	1	1
1	0	0	0	0	0	1
1	1	1	1	1	1	1
0	0	1	0	0	0	0
1	1	0	1	0	0	1
1	0	1	0	1	0	0
0	1	1	0	0	1	0

Pattern-Based Classification on Graphs



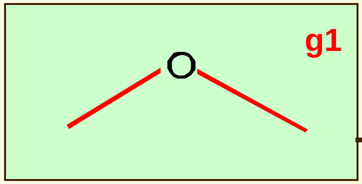


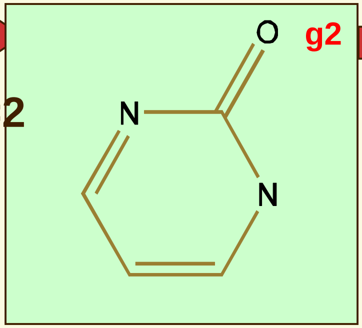


Mining

min_sup=2

Frequent Graphs





Transform

g1	g2	Class
1	1	0
0	0	1
1	1	0

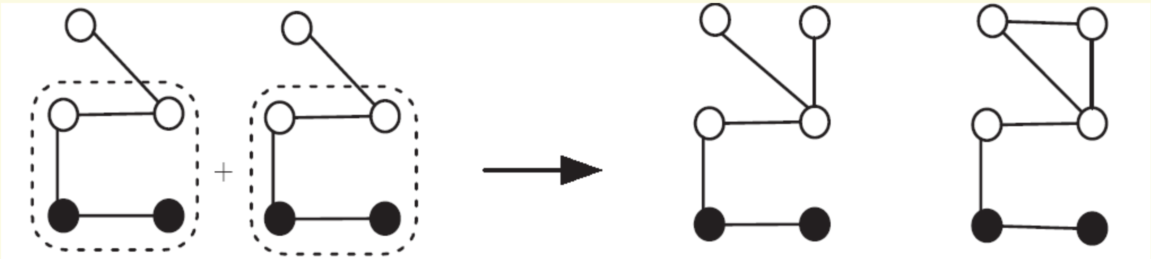
Substructure-Based Graph Classification

- **Basic idea**
  - Extract graph substructures $F = \{g_{1,...,} g_n\}$
  - Represent a graph with a feature vector $\mathbf{x} = \{x_1, ..., x_n\}$
    - where x_i is the frequency of g_i in that graph
  - Build a classification model

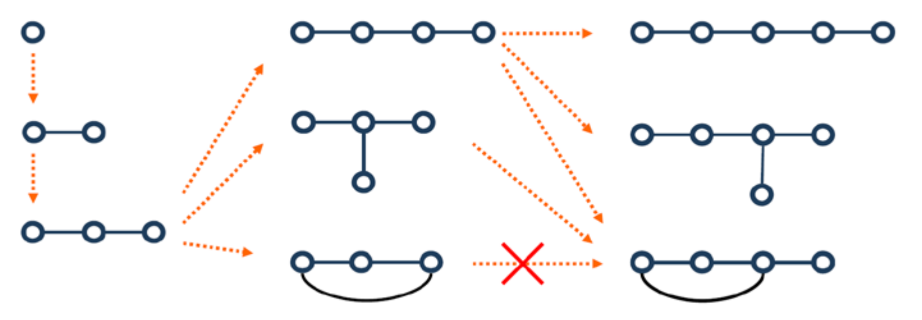
- **Different features and representative work**
  - Fingerprint
  - Maccs keys
  - Tree and cyclic patterns [Horvath et al.]
  - Minimal contrast subgraph [Ting and Bailey]
  - Frequent subgraphs [Deshpande et al.; Liu et al.]
  - Graph fragments [Wale and Karypis]

Recall: two basic frequent pattern mining methods

- Apriori: Join two size-k patterns to a size-(k+1) pattern
  - Itemset: {a,b,c} + {a,b,d} → {a,b,c,d}



- Pattern Growth: Depth-first search, grow a size-k pattern to size-(k+1) one by adding one element

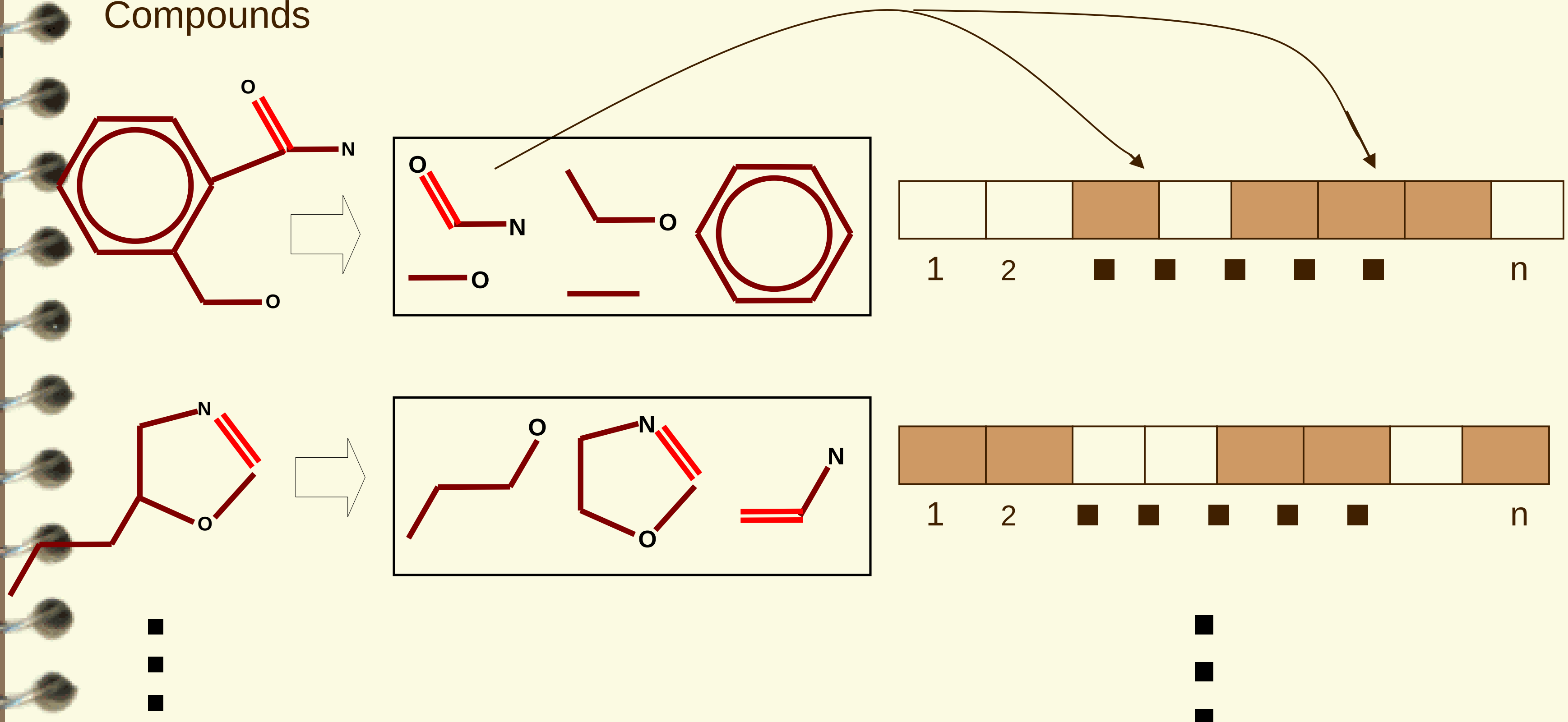


Fingerprints (fp-n)

Chemical Compounds

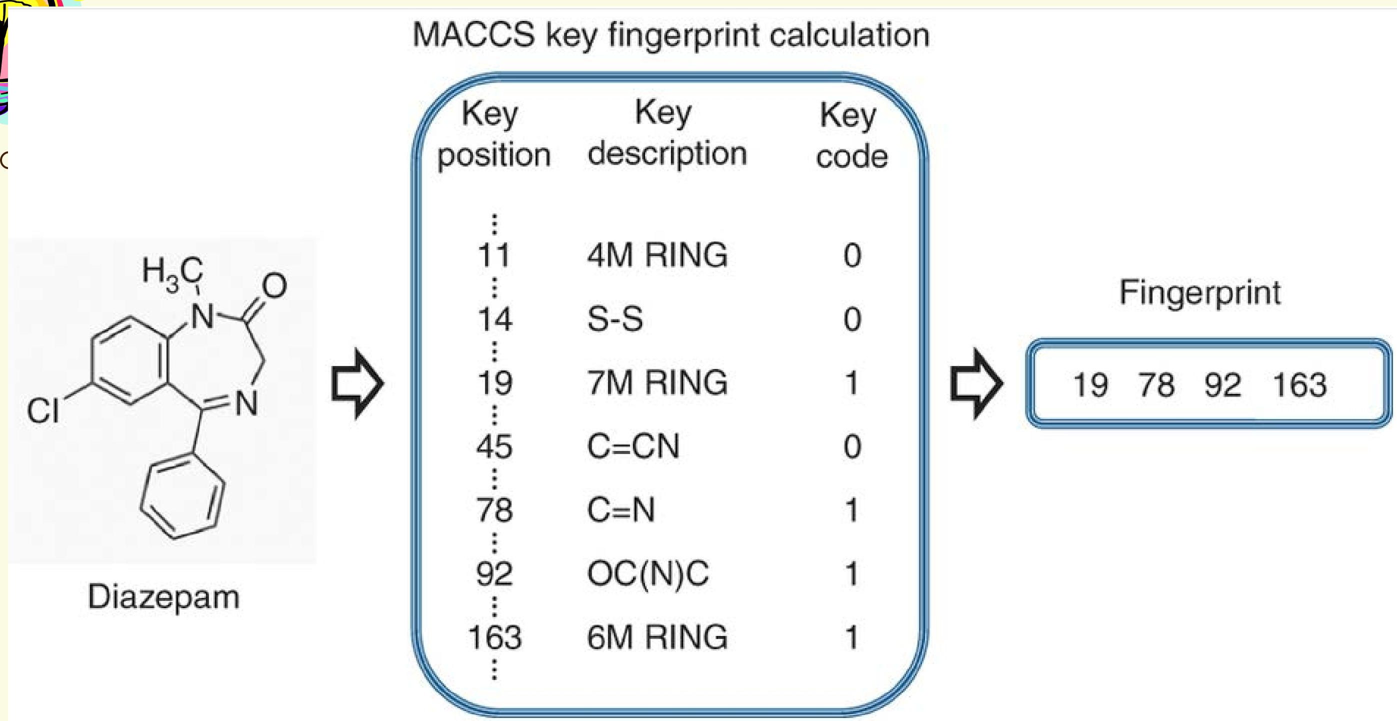
Enumerate all paths up to length *l* and certain cycles

Hash features to position(s) in a fixed length bit-vector



Maccs Keys (MK)

Each Fragment forms a fixed dimension in the

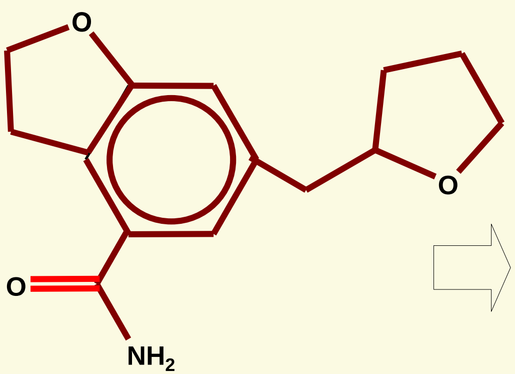


Identify "Important" Fragments
for bioactivity

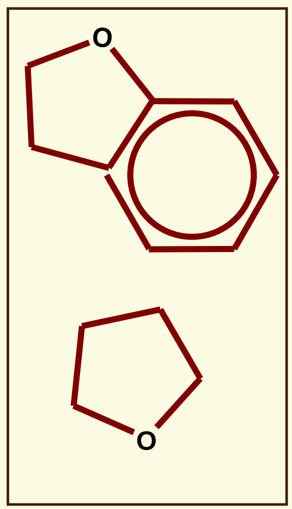
23

Cycles and Trees (CT)
[Horvath et al., KDD'04]

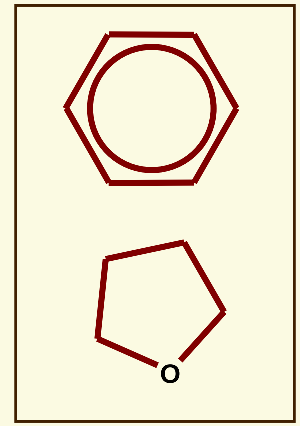
Chemical Compound



Identify
Bi-connected
components

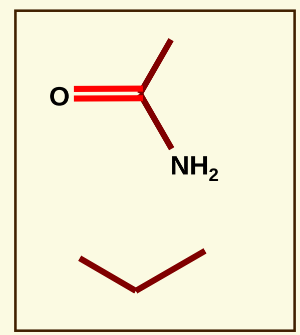


Bounded
Cyclicity
Using
Bi-connected
components



Fixed number
of cycles

Delete
Bi-connected
Components
from the
compound

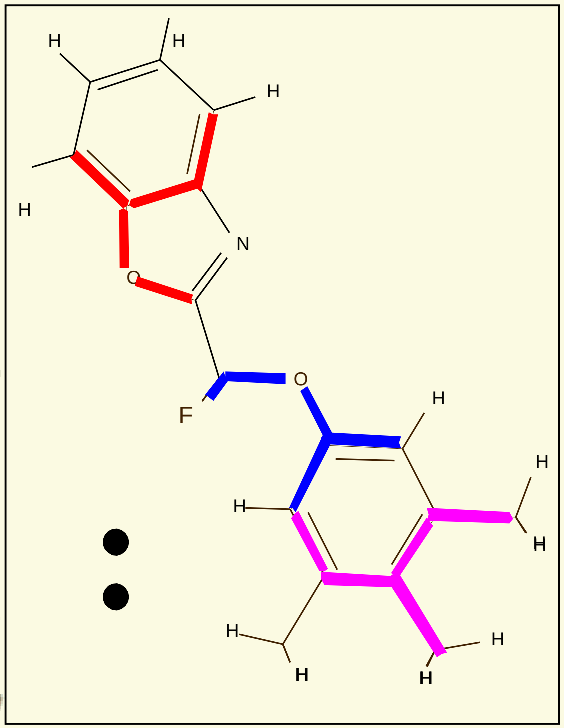


Left-over
Trees

Frequent Subgraphs (FS)
[Deshpande et al., TKDE'05]

Discovering Features

Chemical
Compounds

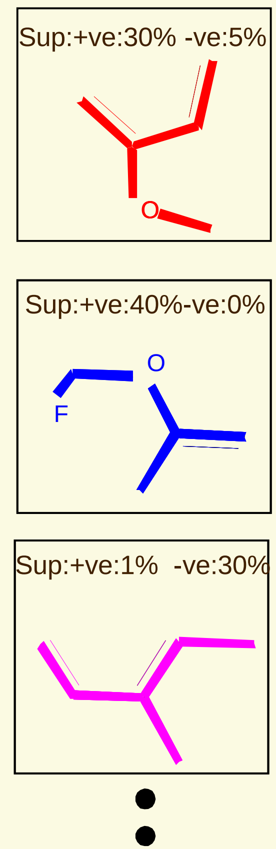


Topological features – captured by graph representation

Discovered
Subgraphs

Frequent
Subgraph
Discovery

Min.
Support.



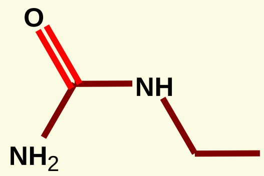
Frequent Subgraph-Based Classification [Deshpande et al., TKDE'05]

- **Frequent subgraphs**
  - A graph is **frequent** if its support (occurrence frequency) in a given dataset is no less than a minimum support threshold
- **Feature generation**
  - Frequent topological subgraphs by FSG
  - Frequent geometric subgraphs with 3D shape information
- **Feature selection**
  - Sequential covering paradigm
- **Classification**
  - Use SVM to learn a classifier based on feature vectors
  - Assign different misclassification costs for different classes to address skewed class distribution

http://www.kernel-machines.org/

Graph Fragments (GF)
[Wale and Karypis, ICDM’06]

- Tree Fragments (TF): At least one node of the tree fragment has a degree greater than 2 (no cycles).



- Path Fragments (PF): All nodes have degree less than or equal to 2 but does not include cycles.

- Acyclic Fragments (AF): TF U PF
  - Acyclic fragments are also termed as free trees.

Graph Fragment (GF) [Wale and Karypis, ICDM'06]

- **All graph substructures up to a given length (size or # of bonds)**
  - Determined dynamically → Dataset dependent descriptor space
  - Complete coverage → Descriptors for every compound
  - Precise representation → One to one mapping
  - Complex fragments → Arbitrary topology

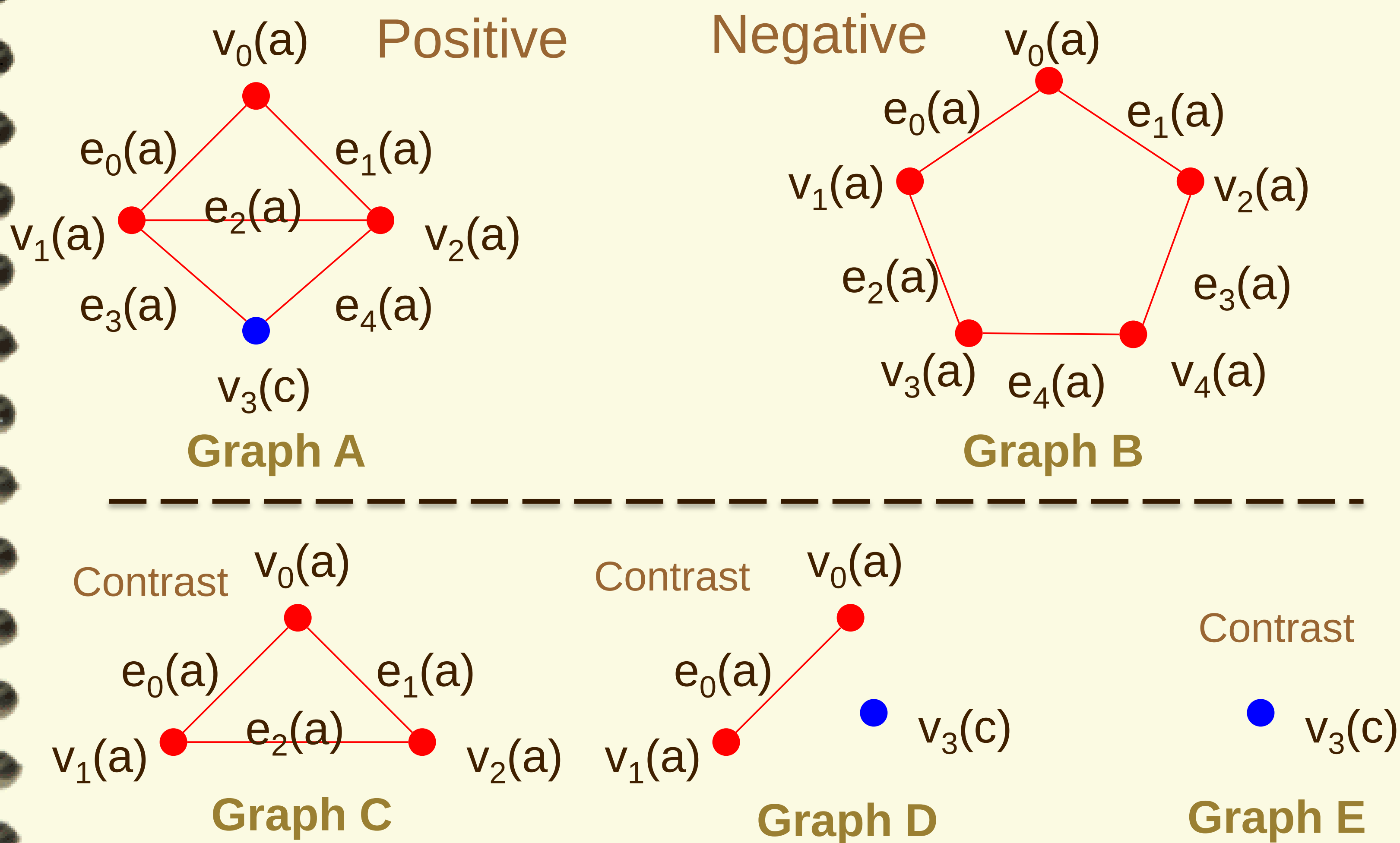
- **Recurrence relation to generate graph fragments of length l**

$$F(G,l) = \begin{cases} \emptyset, & \text{if } G \text{ has fewer than } l \text{ edges or } l = 0 \\ eF(G\backslash e, l-1) \cup F(G\backslash e, l), & \text{otherwise,} \end{cases}$$

Minimal Contrast Subgraphs [Ting and Bailey, SDM'06]

- A contrast graph is a subgraph appearing in one class of graphs and never in another class of graphs
  - Minimal if none of its subgraphs are contrasts
  - **May be disconnected**
    - Allows succinct description of differences
    - But requires larger search space
- Main idea
  - Find the maximal common edge sets
    - These may be disconnected
  - Apply a minimal hypergraph transversal operation to derive the minimal contrast edge sets from the maximal common edge sets
  - Must compute minimal contrast vertex sets separately and then minimal union with the minimal contrast edge sets

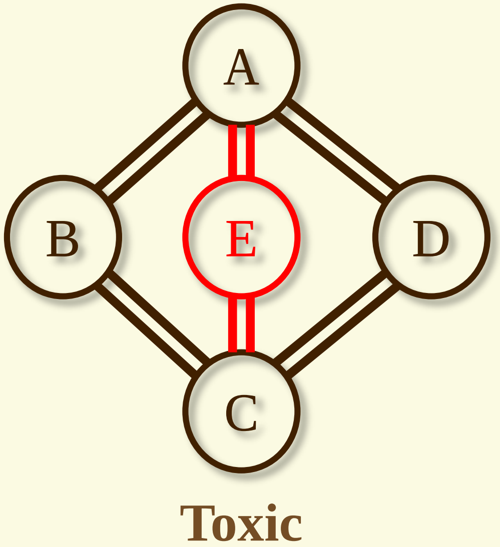
Contrast subgraph example



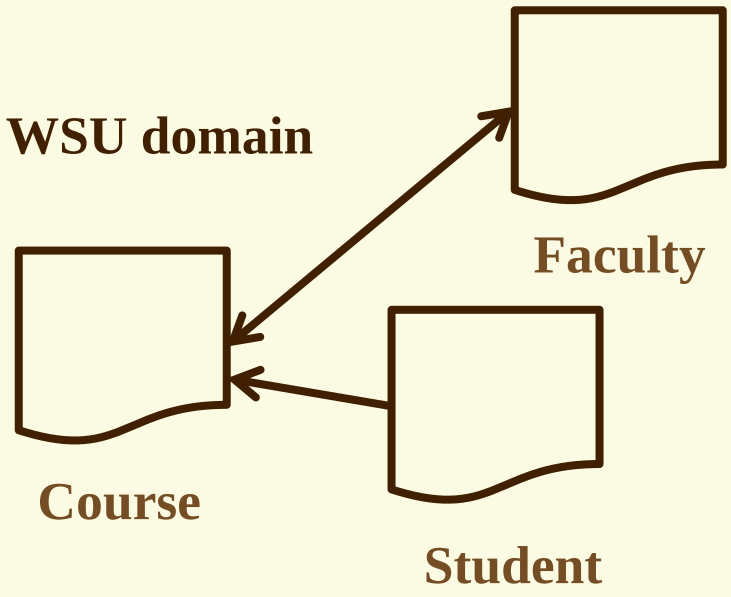
Kernel based Graph Classification

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



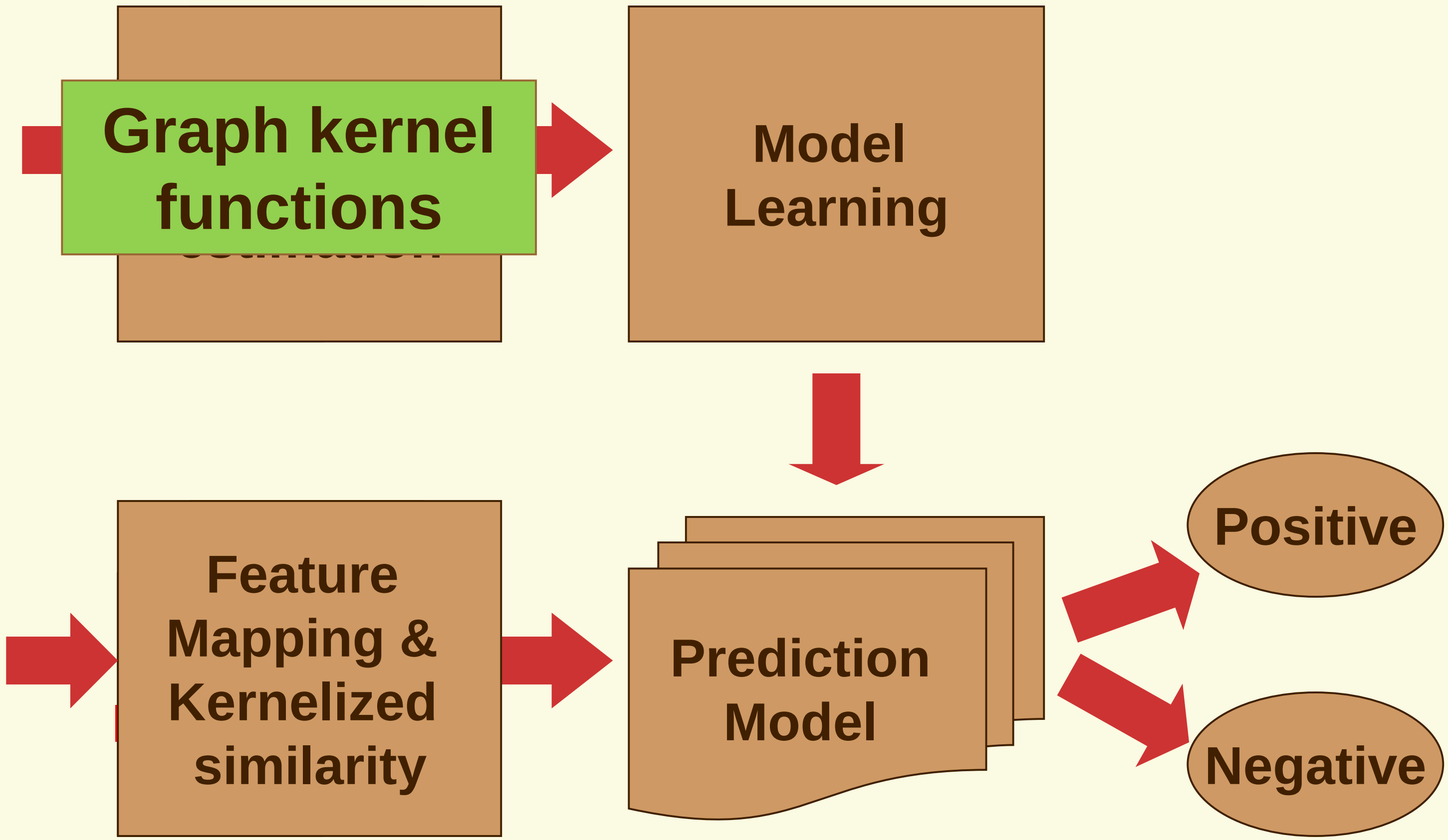
Relating Graph Structures to Classes?

- ✓ Two step process:
  - Devise kernel that captures property of interest
  - Apply *kernelized* classification algorithm, using the kernel function.
- ✓ Graph kernels looked at
  - Classification of Graphs
    - Direct Product Kernel
  - Classification of Vertices
    - Laplacian Kernel
- ✓ *With support vector machines* (SVM), one of the more well-known kernelized classification techniques.

Graph Kernels

- Kernel is a type of similarity function
  - $K(x, y)>0$ is the “similarity” of x and y
  - a feature representation f can define a kernel: $f(x) = (f1(x), f2(x)\ldots fk(x))$
    - $K(x,y) = f(x)f(y) = \text{sum } f\_i(x) f\_i(y)$
- Motivation:
  - Kernel based learning methods doesn’t need to access data points
    - rely on the kernel function between the data points
  - Can be applied to any complex structure provided a kernel function on them
- Basic idea:
  - Map each graph to some significant set of patterns
  - Define a kernel on the corresponding sets of patterns

Graph Kernel-Based Classification



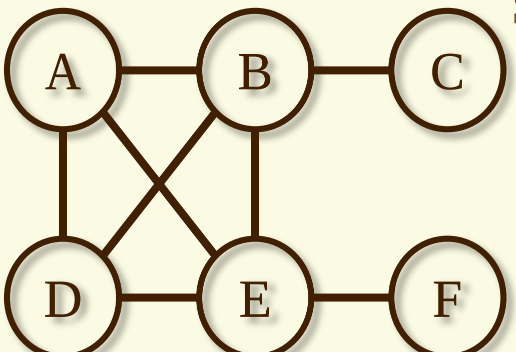
More features or more training examples?

36

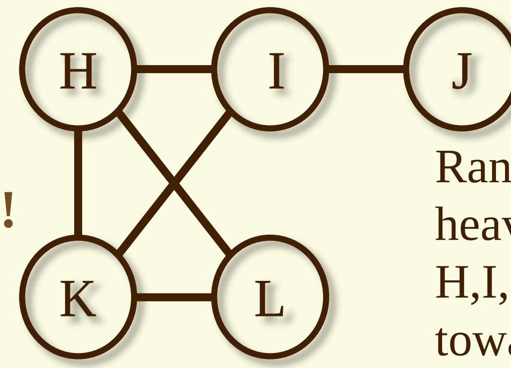
Walk-based kernels (RW Kernel)

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

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

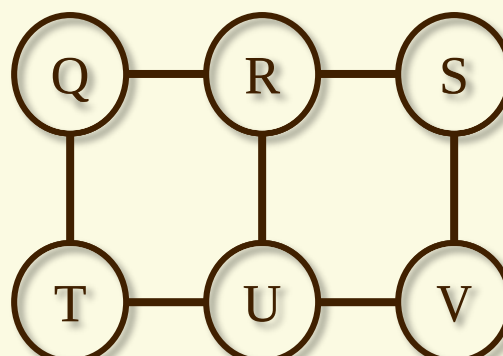


Similar!



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

Not Similar!



Random walk vertices evenly distributed

Random walk kernel

- ✓ Basic Idea: count the matching random walks between the two graphs
- ✓ Marginalized Kernels (Gärtner '02, Kashima et al. '02, Mahé et al.'04)

$$K(G_1, G_2) = \sum_{h_1} \sum_{h_2} p(h_1) p(h_2) K_L(l(h_1), l(h_2))$$

  - h_1 and h_2 are paths in graphs G_1 and G_2
  - $p(h_1)$ and $p(h_2)$ are probability distributions on paths
  - $K_L(l(h_1), l(h_2))$ is a kernel between paths, e.g.,

$$K_L(l_1, l_2) = \begin{cases} 1 & \text{if } l_1 = l_2, \\ 0 & otherwise. \end{cases}$$

Direct Product Kernel

Input Graphs

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

Direct Product

$$G_X = G_1 \times G_2$$

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

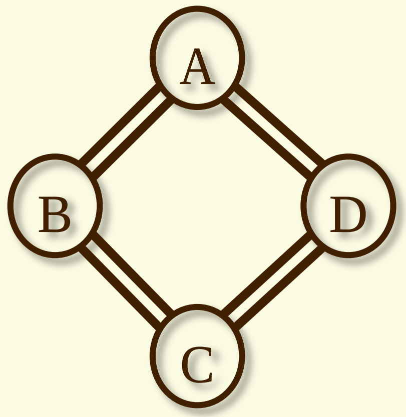
Direct Product Vertices

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

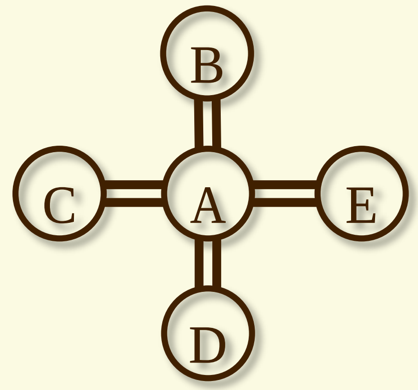
Direct Product Edges

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

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 / Type-B		A					B					C					D				
		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

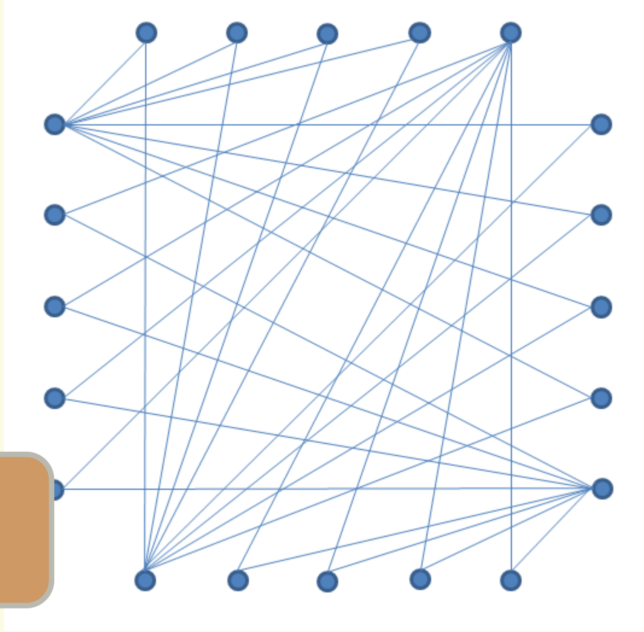
1. Compute direct product graph $\boldsymbol{G_x}$
2. Compute the maximum in- and out-degrees of *Gx*, *di* and *do*.
3. Compute the decay constant

indexed by all possible walks

4. Compute the infinite weighted geometric series of walks (array *A*).
5. Su

common walks on G1 and G2 are the walks of G1xG2

$$k_{\times(g_1,g_2)} = \sum_{i,j=1}^{|V_{g_1\times g_2}|} \left[\sum_{\ell=0}^{\infty} \lambda_\ell M_{g_1\times g_2}^{\ell} \right]_{ij}$$



Direct Product Graph of Type-A and Type-B

Kernel Matrix

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

- **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**

Summary

- Basics in data mining: what are several basic tasks in data mining? Comparing with statistics? Databases? A wide range of interesting applications (pattern recognition, business rule mining, scientific data…)
- Graph mining: graph pattern mining, classification, clustering
- Frequent graph pattern mining
  - Apriori methods
  - Pattern growth
- Graph classification
  - Feature-based
  - Kernel-based
  - Model learning: decision trees, SVM…

Related techniques

- Graph mining:
  - Frequent Subgraph
  - Mining Anomaly Detection
  - Kernel

    –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).
- Tree kernels – similarity of graphs that are trees.

Related techniques

- ✓ 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.
  - Support kernel functions working on vectors.

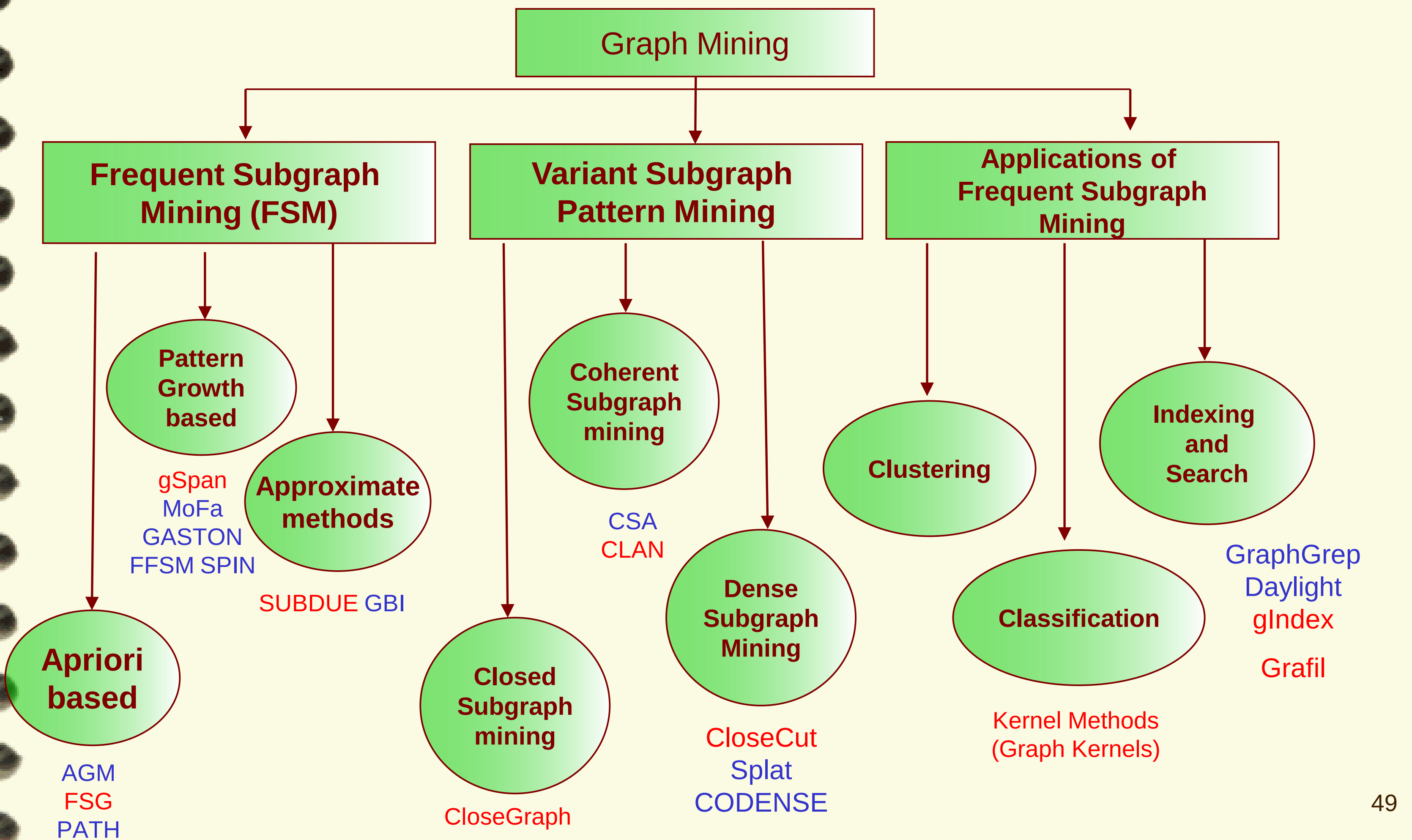
Related – Ensemble Classification

- 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 techniques – Evaluating, Comparing Classifiers

- ✓ 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)

A big picture



Papers to review

- Yan, Xifeng, and Jiawei Han. "gspan: Graph-based substructure pattern mining." *Data Mining, 2002. ICDM 2003.*
- Inokuchi, Akihiro, Takashi Washio, and Hiroshi Motoda. "An apriori-based algorithm for mining frequent substructures from graph data." *Principles of Data Mining and Knowledge Discovery*. Springer Berlin Heidelberg, 2000. 13-23.
- Yan, Xifeng, and Jiawei Han. "CloseGraph: mining closed frequent graph patterns." *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003.
- Kudo, Taku, Eisaku Maeda, and Yuji Matsumoto. "An application of boosting to graph classification." *Advances in neural information processing systems*. 2004.
- Kashima, Hisashi, and Akihiro Inokuchi. "Kernels for graph classification."*ICDM Workshop on Active Mining*. Vol. 2002. 2002.
- Saigo, Hiroto, et al. "gBoost: a mathematical programming approach to graph classification and regression." *Machine Learning* 75.1 (2009): 69-89.
- Horváth, Tamás, Thomas Gärtner, and Stefan Wrobel. "Cyclic pattern kernels for predictive graph mining." *KDD*, 2004.
- Ting, Roger Ming Hieng, and James Bailey. "Mining Minimal Contrast Subgraph Patterns." *SDM*. 2006.