Node Classification

Prof. O-Joun Lee

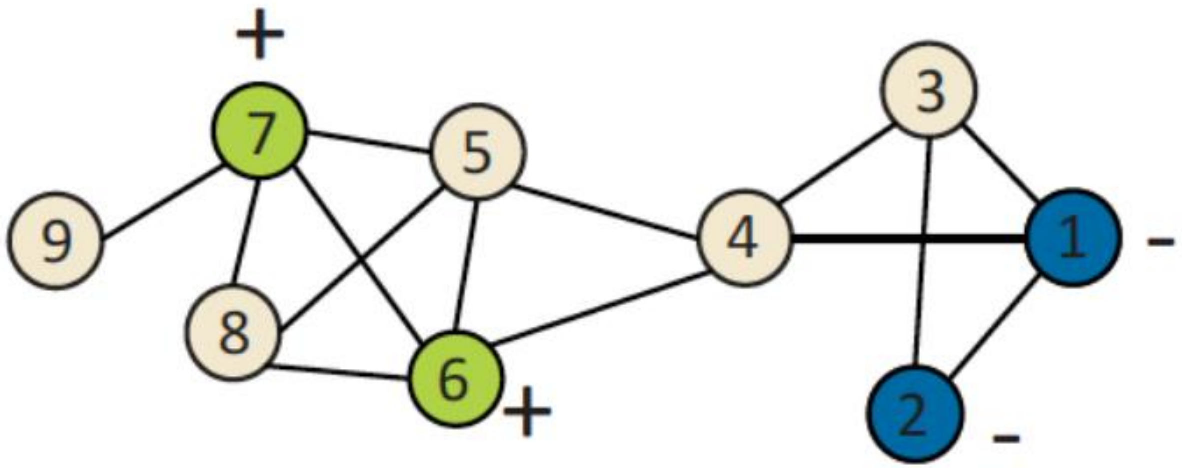
Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

네트워크 과학 연구실 NETWORK SCIENCE LAB
가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

Contents

Classification with Network Data

- Given a graph and few nodes for which we know the "label" or a "class," how can we predict user attributes or interests?



Predict the labels for non marked nodes?

Why node classification?

- Is this a friend or an aquitance?
- Recommendation systems to suggest objects (music, movies, activities)
- Automatically understand roles in a network (hubs, activators, influencing nodes, etc.)
- Identify experts for question answering systems
- Targeted advertising
- Study of communities (key individuals, group starters ...)
- Study of diseases and cures
- Identify unusual behaviours or behavioural changes
- Finding similar nodes and outliers

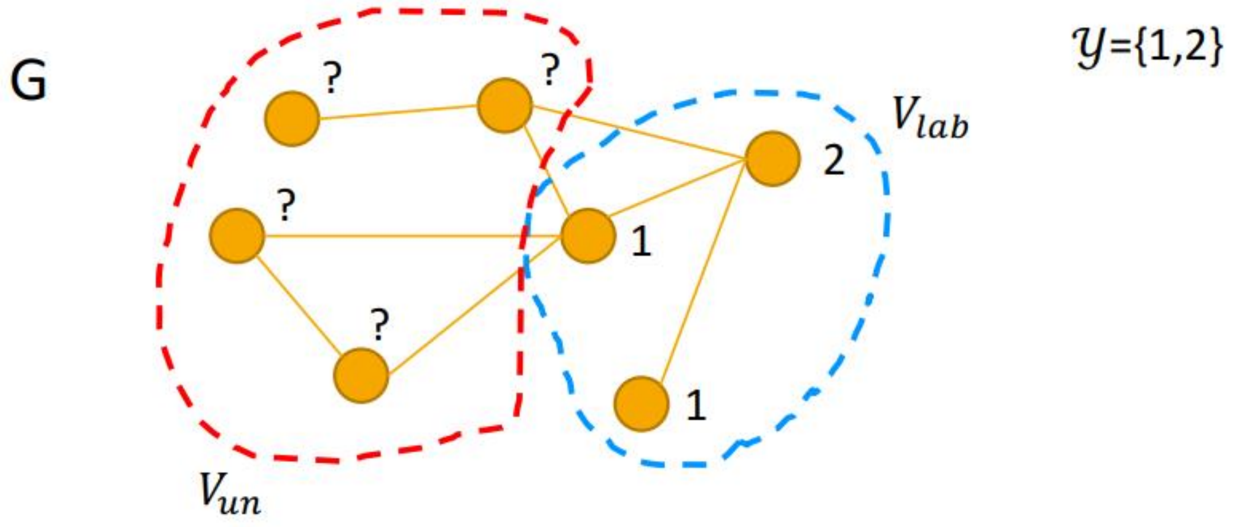
Why node classification is useful?

- Not all the nodes have labels
  - users are not willing to provide explanations
- Roles are not explicitly declared
  - who is more important in a company? (Think about the exchanged emails)
- Labels provided by the users can be misleading
- Labels are sparse
  - some categories might be missing or incomplete

Node classification problem

- Given:
  - Graph G =(V, E, W) with vertices V, edges E and weight matrix W
  - Labeled nodes V_{lab} ⊂ V, unlabeled nodes V_{un} = V \ V_{lab}
  - Y the set of m possible labels (e.g., Y={republican, democrat})
  - Y_{lab} = $\{y_1, y_2,$ …. , $y_l\}$ the labels on labeled nodes in V_{lab}

- Problem:
  - Infer labels Y_{un} for all nodes in V_{un}

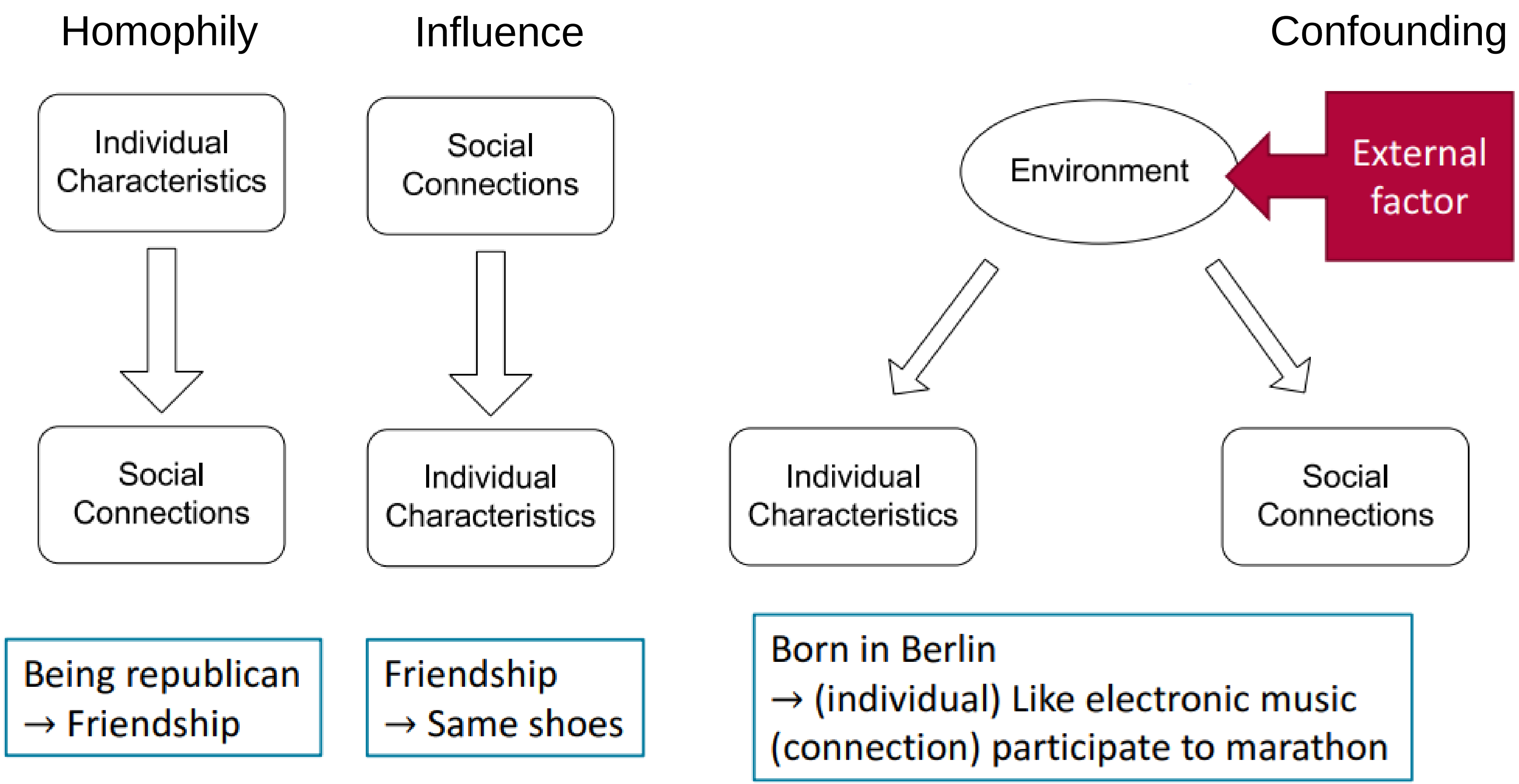


Node classification problem (2)

- Can be generalized to multilabel and multiclass classification:
  - With multiclass classification assume that each labelled node has a probability distribution on the labels.
- Can work on generalized graph structures
  - hypergraphs, graphs with weighted, labelled, timestamped edges, multigraphs, probabilistic graphs and so on.

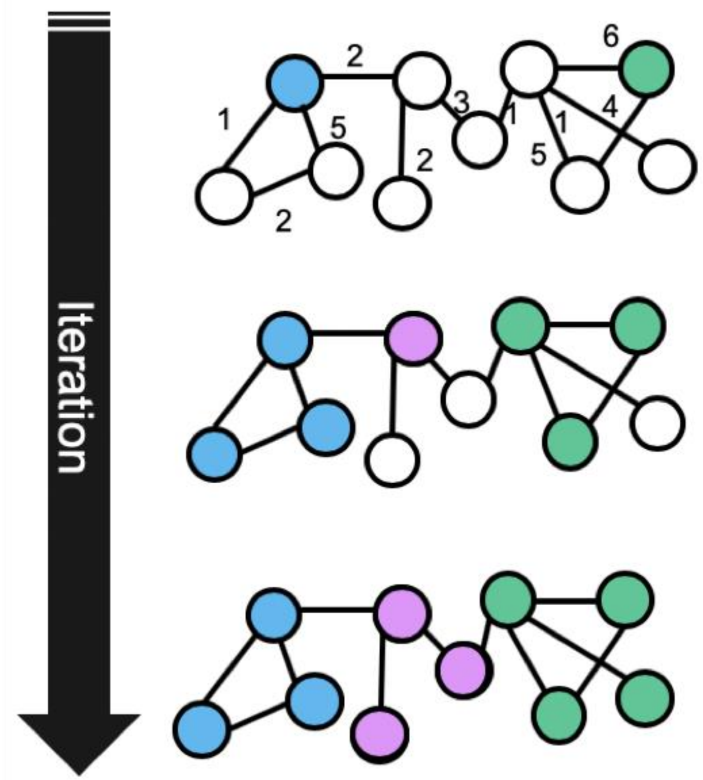
Influential factors on Networks

- Individual behaviours are correlated in a network environment



The importance of the graph structure

- The graph structure encodes important information for node classification
- So, it is reasonable to think that:
  - labels propagate in the network following the links
- Methods that work with points in the space perform poorly in a graph



Assumption: The label propagates on the network

Node features

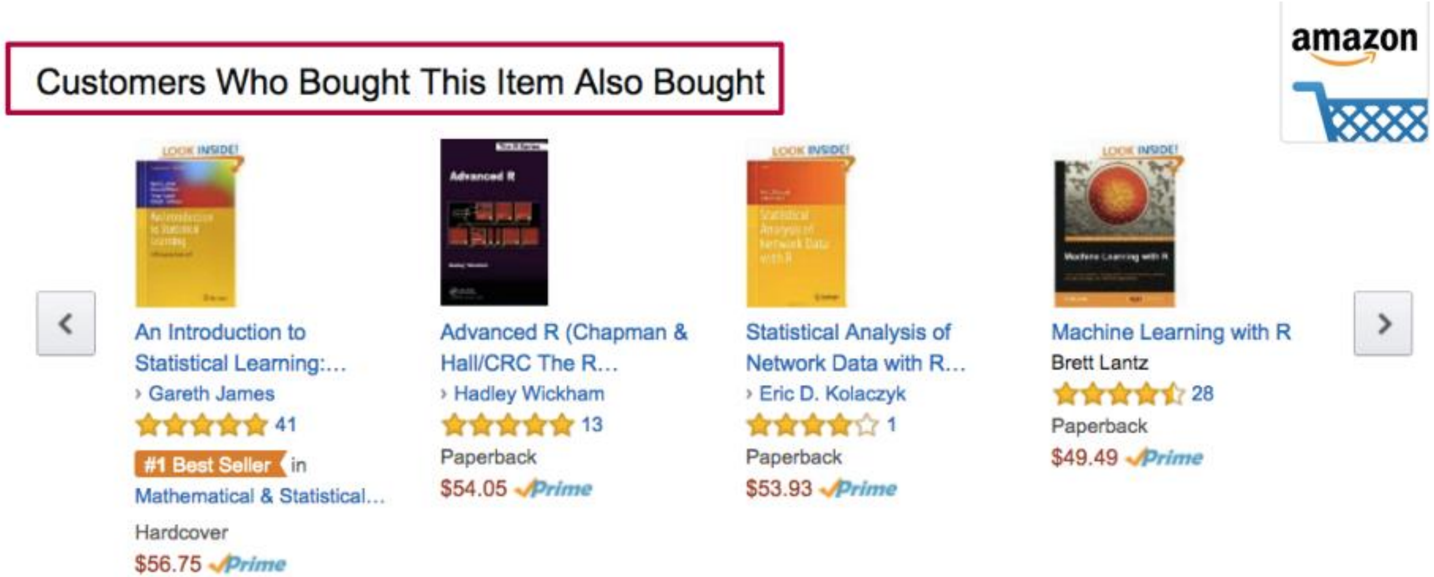
- Node features:
  - Measurable characteristics of the nodes that help
    - discriminating a node from another
    - or stating the similarity with other nodes.

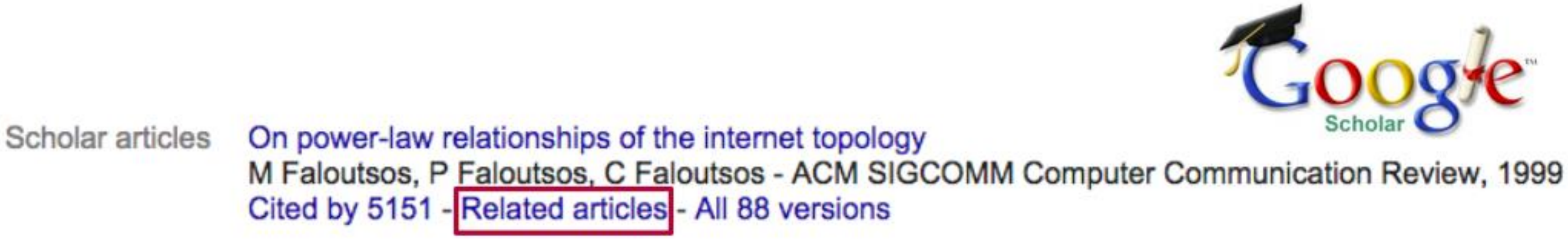
- Examples of features:
  - In/out degree of the node
  - Number of L-labelled edges from that node
  - Number of paths in that goes through the node
  - Number of triangles
  - Degree and number within ego-net edges
  - etc.

- **Similarity based**
  - Find nodes that share the same characteristics with other nodes
- **Iterative learning**
  - Learn a set of labels and propagate the information to similar nodes
- **Label propagation**
  - Labelled nodes propagate the information to the neighbours with some probability

Similarity based

- Real-world Applications:



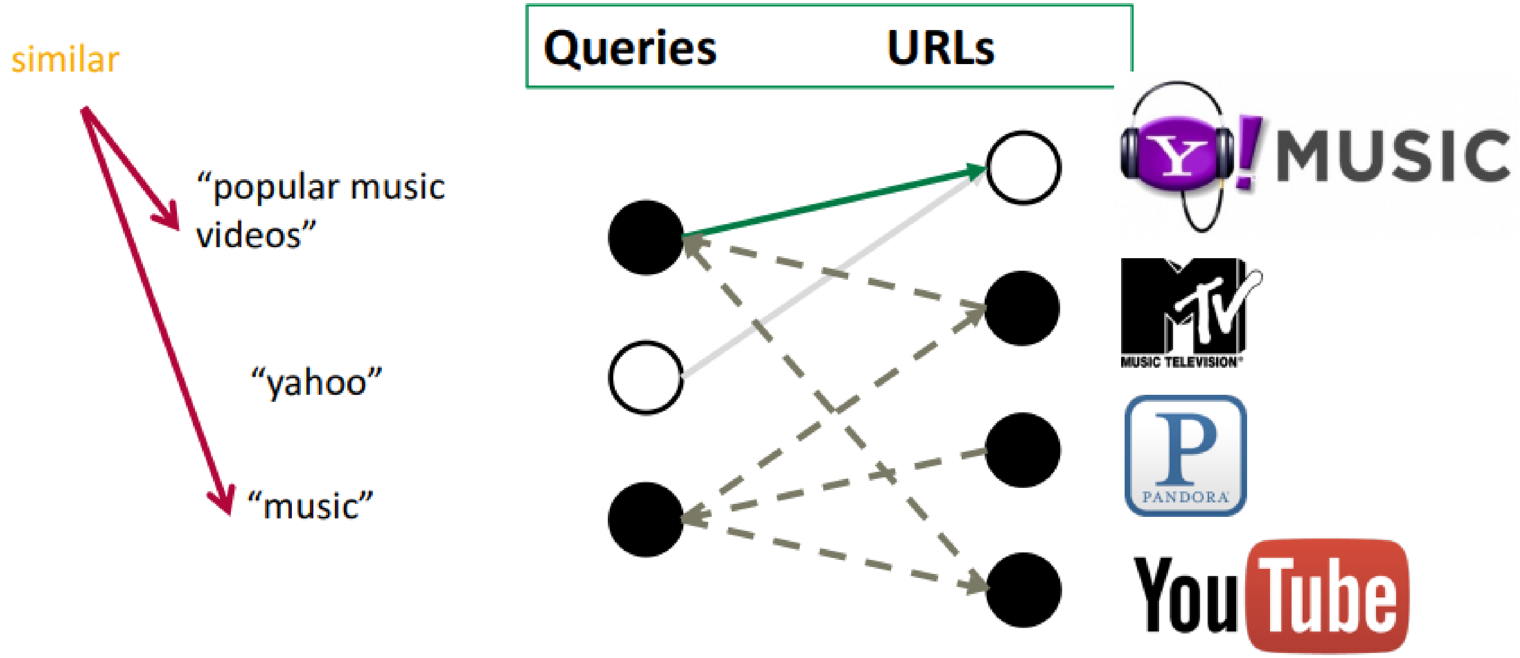


Similarity based

- Movies recommendations

Search Engines

- Topical search engine is an engine that focuses on a particular topic.
- It covers a part of the whole Web rather than a particular website - this is possible because Programmable Search Engine allows you to include multiple websites in the same engine.



Similarity based approaches

- Equivalences in terms of structure
  - Structural, Automorphic, and Regular
- Role extraction methods:
  - RolX
- Recursive similarities
  - Paths, Max-flow, SimRank

History: Equivalences

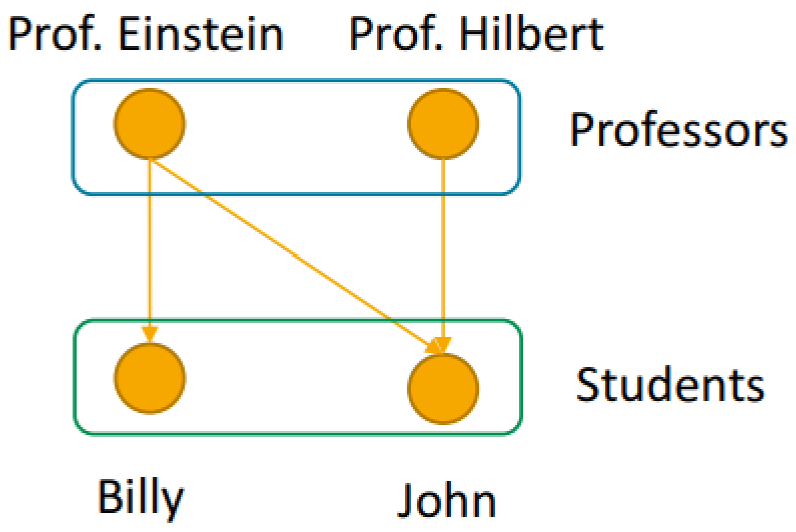
- Two nodes u and v are structurally equivalent if they have the same relationships to all other nodes.

- Two nodes u and v are automorphically equivalent if all the nodes can be relabelled to form an isomorphic graph with the labels of u and v interchanged (just change the node id).

- Two nodes u and v are regularly equivalent if they are equally related to equivalent others

Regular equivalence

- Two nodes u and v are regularly equivalent if they are equally related to equivalent others
- Assumes a similarity between sets of nodes

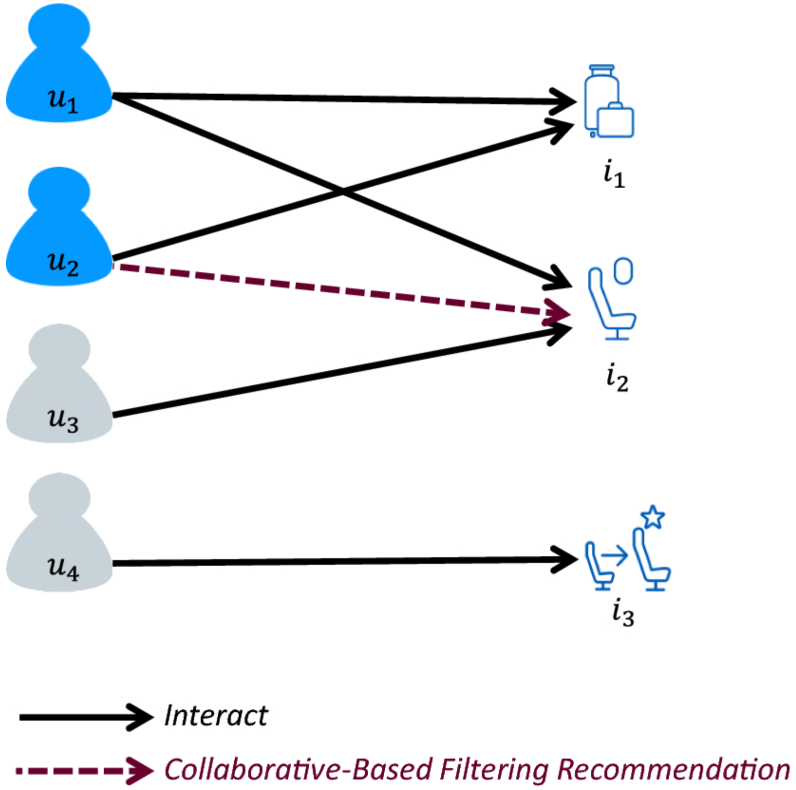


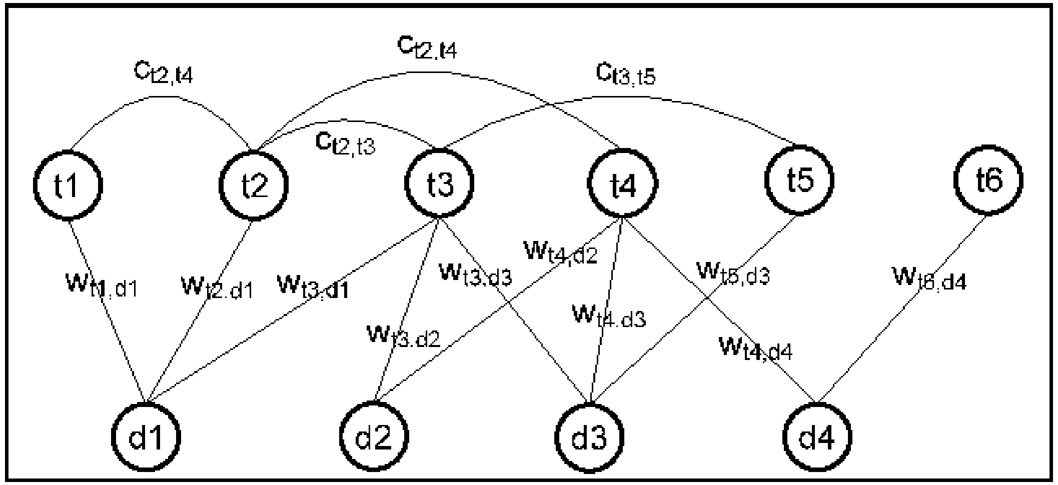
Billy and John are similar because they are both connected to a professor.
Same for prof. Einstein and Hilbert.

Regular equivalence doesn't care about which connections but to which set/group a node is connected

Regular equivalence

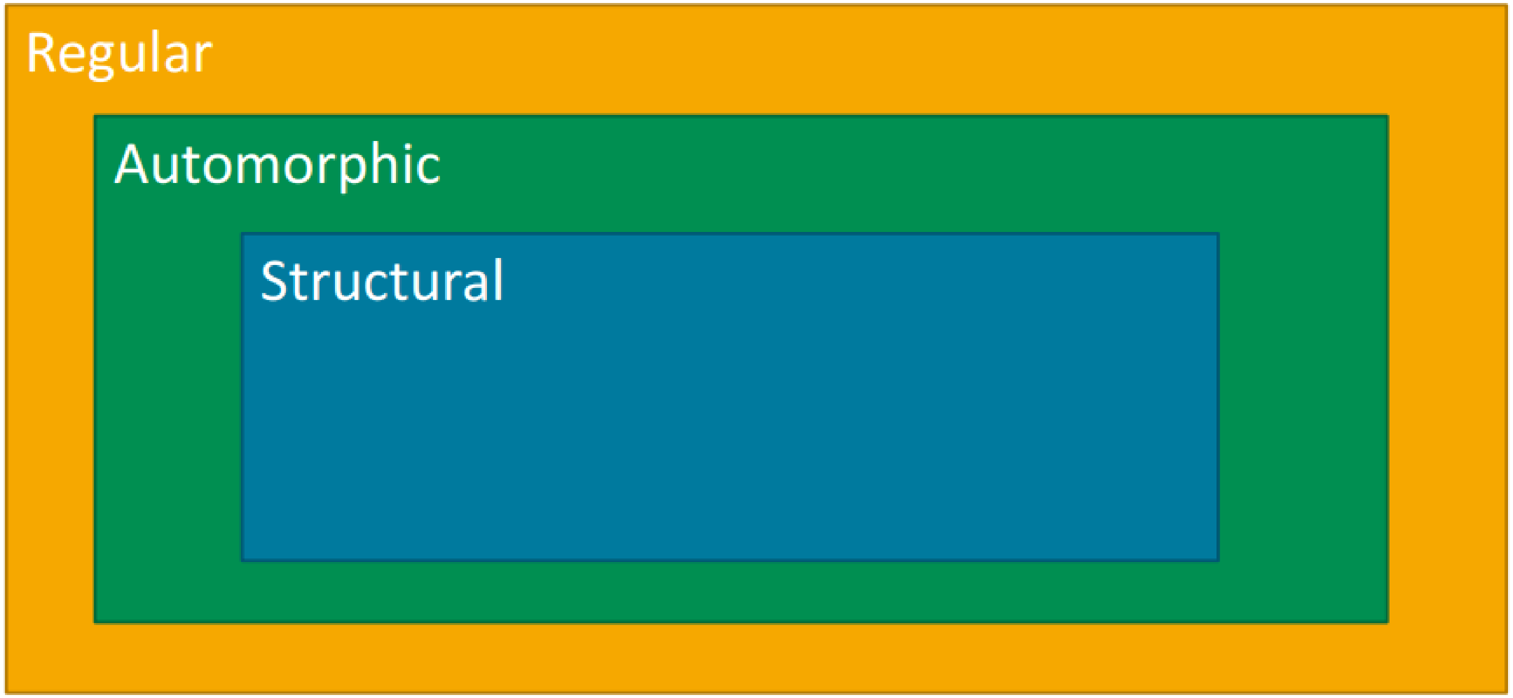
- Two nodes u and v are regularly equivalent if they are equally related to equivalent others
- Assumes a similarity between sets of nodes





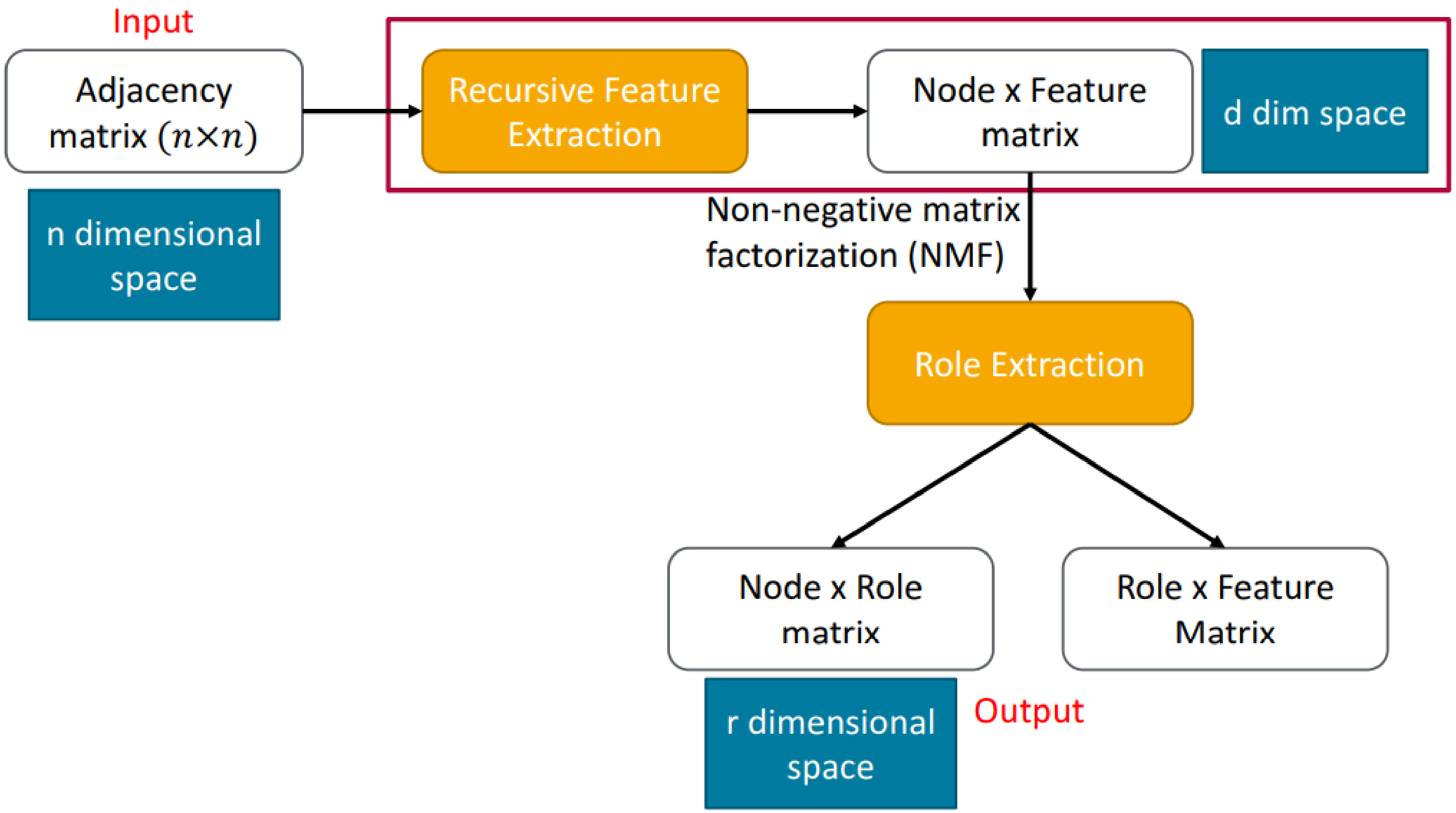
Relation among equivalences

- What is the relation among the three equivalences?



RolX: Role eXtraction algorithm

- Takes features extracted with ReFeX and factorizes the binary node-feature matrix in order to create low dimensional structural node representations

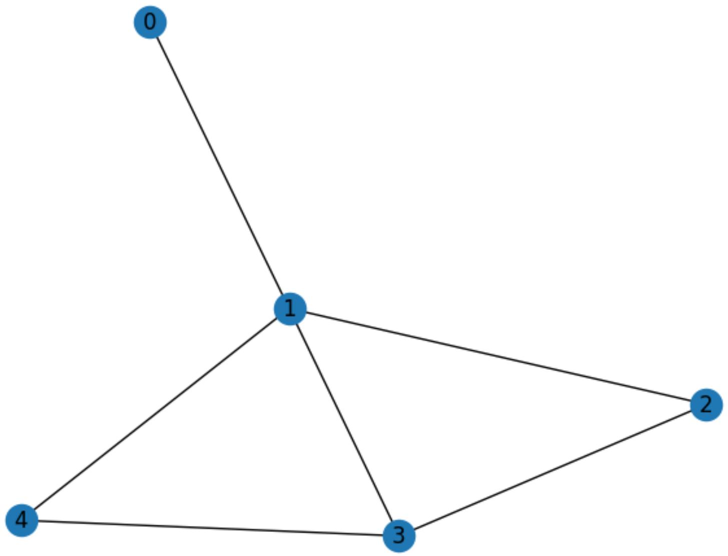


Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C. and Li, L., 2012. Rolx: structural role extraction & mining in large graphs. SIGKDD

RolX: Role eXtraction algorithm: Sample code


```
# assign node roles
role_extractor = RoleExtractor(n_roles=None)
role_extractor.extract_role_factors(features)
node_roles = role_extractor.roles

print('\nNode role assignments:')
pprint(node_roles)

print('\nNode role membership by percentage:')
print(role_extractor.role_percentage.round(2))
```

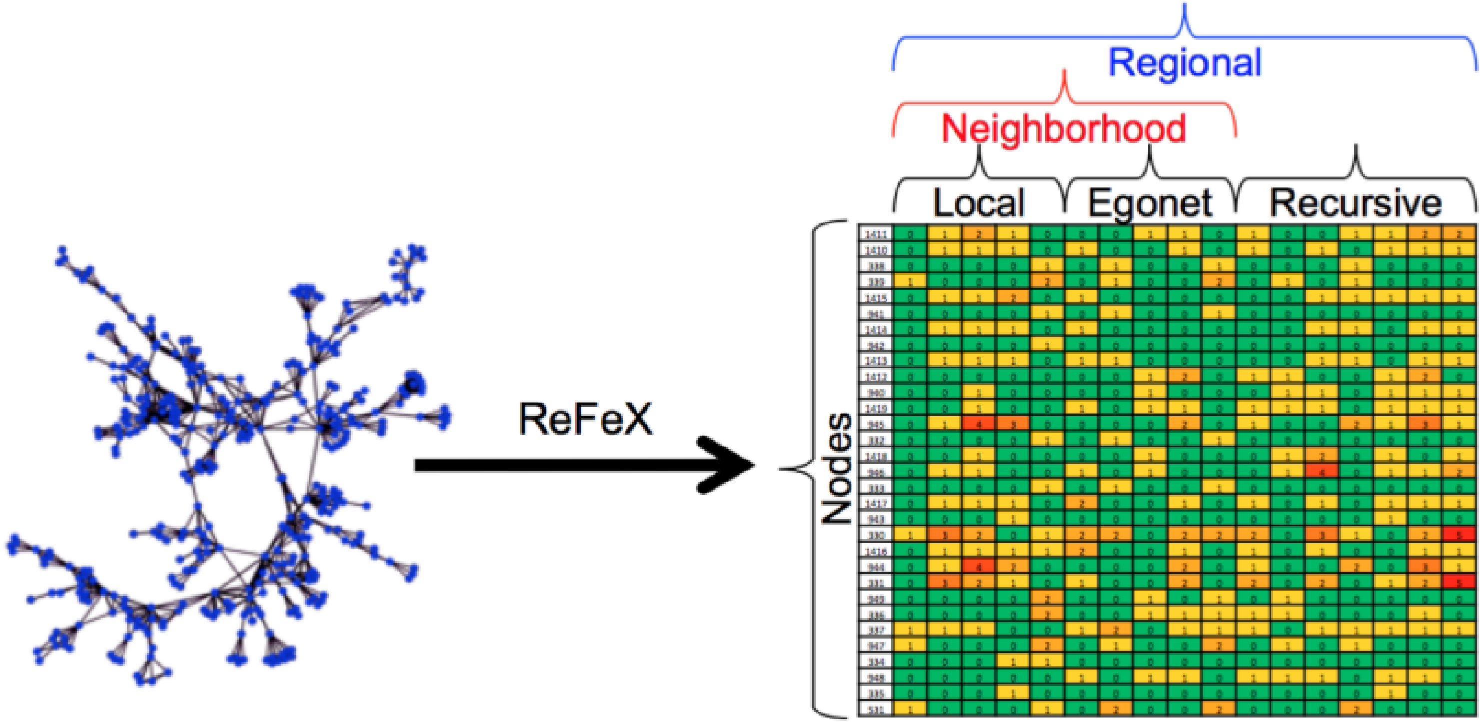
```
Node role assignments:
{0: 'role_1', 1: 'role_0', 2: 'role_1', 3: 'role_0', 4: 'role_1'}

Node role membership by percentage:
   role_0  role_1
0    0.03    0.97
1    0.97    0.03
2    0.25    0.75
3    0.69    0.31
4    0.25    0.75
```



Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C. and Li, L., 2012. Rolx: structural role extraction & mining in large graphs. SIGKDD

Recursive Feature extraction (ReFex)

- Transform the network connectivity into recursive structural features.
- Technically, embeds the graph into an $|\mathcal{F}|$ dimensional space, where $\mathcal{F}$ is a set of features (degree, self-loops, avg edge weight, # of edges in egonet)



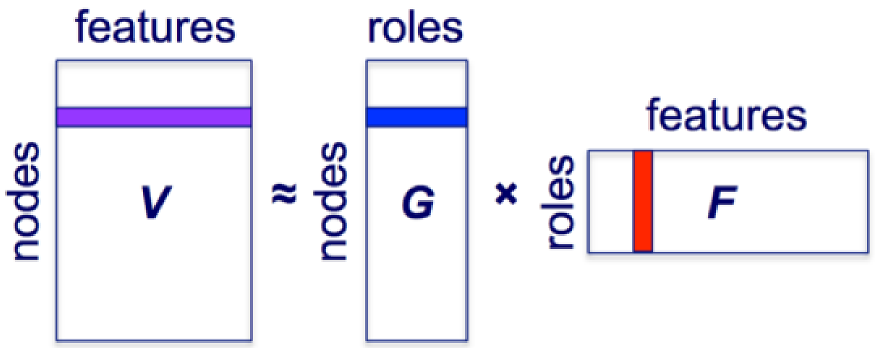
Henderson, K., Gallagher, B., Li, L., Akoglu, L., Eliassi-Rad, T., Tong, H. and Faloutsos, C., 2011. It's who you know: graph mining using recursive structural features. SIGKDD.

ReFeX: Mining features

- Local:
  - Measures of the node degree
- Egonet:
  - The egonet (or ego-network) of a node is the node itself, the adjacent nodes, and the graph induced by those nodes
  - Computed based on each node's ego network: #of within-egonet edges, #of edges entering & leaving the egonet
- Recursive
  - Some aggregate (mean, sum, max, min, etc.) of another feature over a node's neighbours
  - The aggregation can be computed over any real-valued feature, including other recursive features.

Role extraction: Feature grouping

- Find r overlapping clusters in the feature space
  - Each node can have multiple roles at the same time
- Generate a rank r approximation of the node x feature matrix V
- Use non-negative matrix factorization: $V \approx GF$



- The *G* matrix assigns nodes to roles
- The *F* matrix represents how the features explain the roles

SimRank

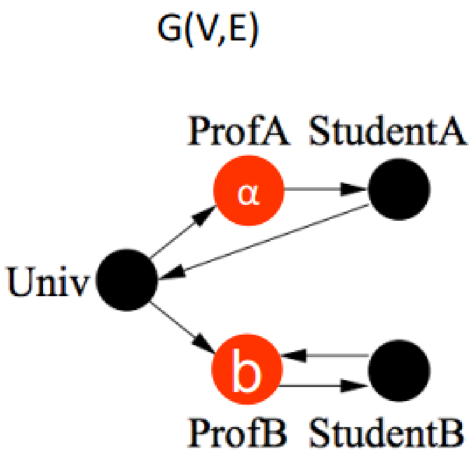
- Idea:
  - Two objects are similar if they are referenced by similar objects

decay factor in [0,1]

$$s(a,b) = \boxed{\text{Avg similarity between in-neighbors of } \alpha \text{ and in-neighbors of } b}$$

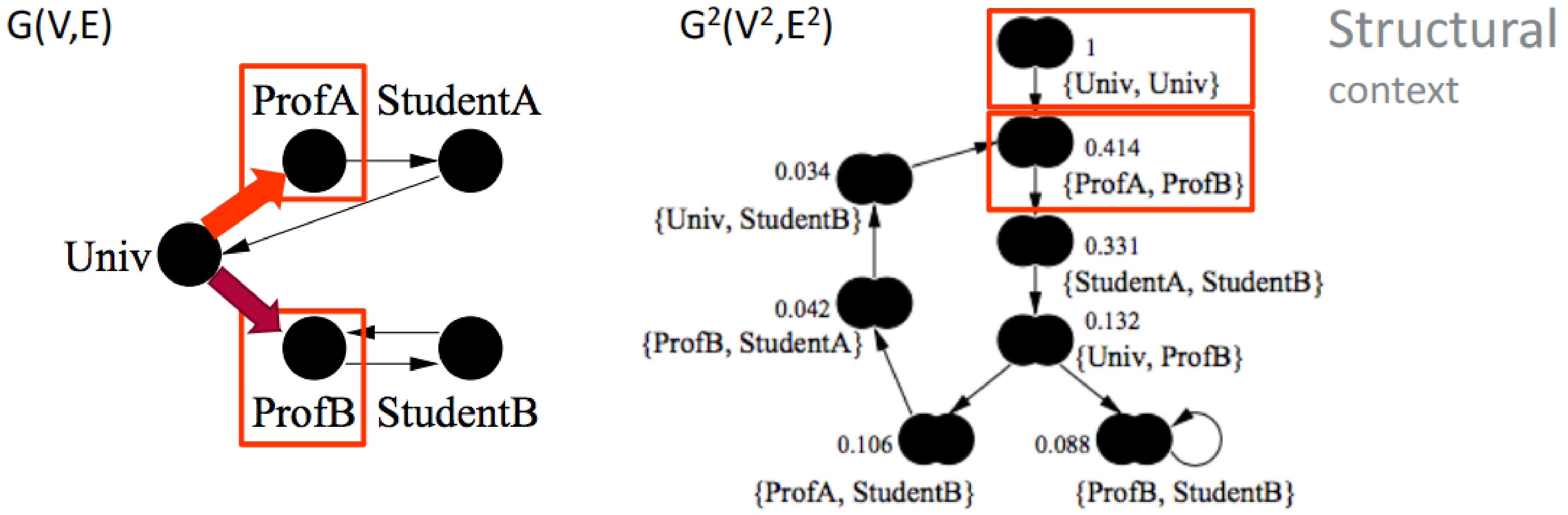
total # of in-neighbors pairs

similarity of in-neighbors



SimRank: Intuition

- Intuition: Computing SimRank is like propagating on the G^2 graph of node-node pairs
  - The source of similarity is self-vertices, like (Univ, Univ).
  - Similarity propagates along pair-paths in G^2, away from the sources.

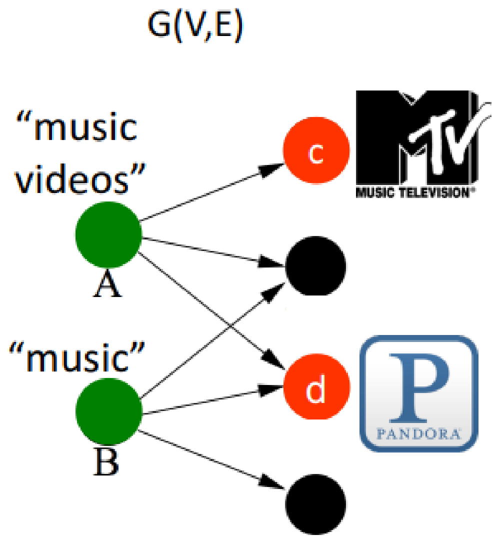


SimRank for Bipartite Graphs

- Average similarity between A and B:

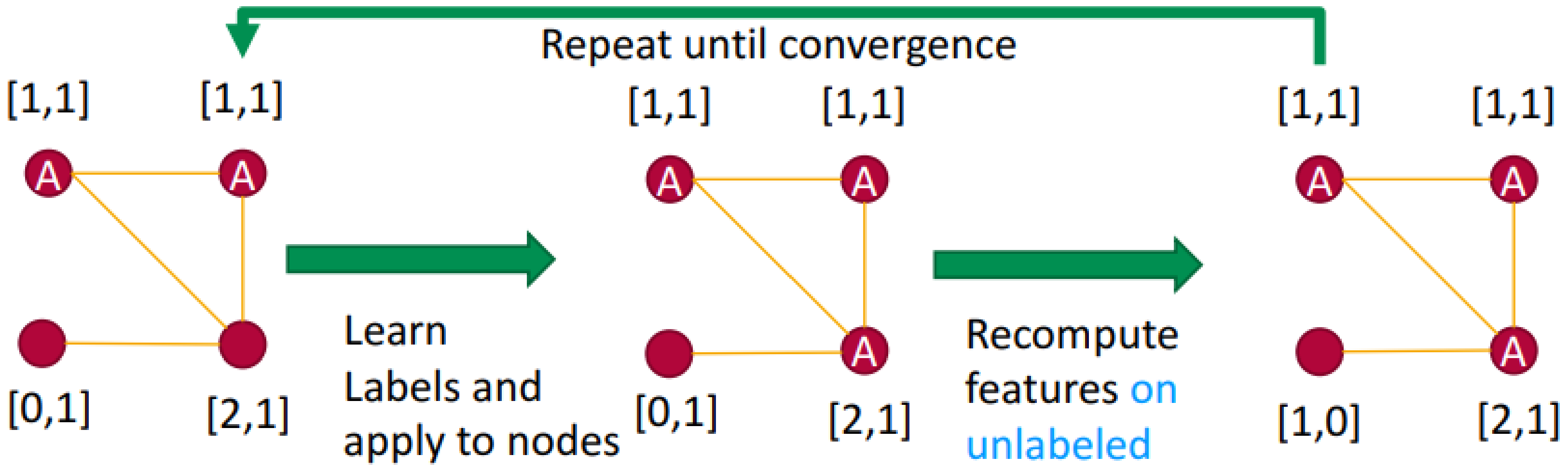
s(A,B) = Avg similarity between out-neighbors of A and out-neighbors of B

s(c,d) = Avg similarity between in-neighbors of c and in-neighbors of d



Iterative Classification Methods

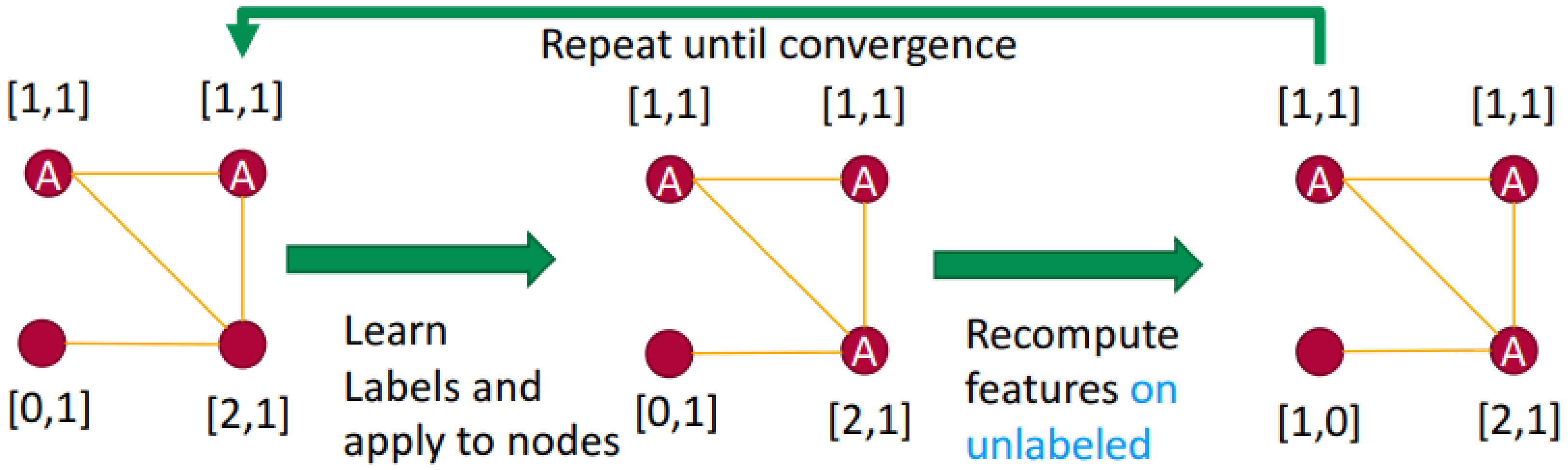
- Idea:
  - Use features that consider the neighbor nodes
  - and repeat the classification several time until nothing changes
- Suppose for each node we have two features: First
  - Number of neighbors with class A
  - Number of neighbors without a class



Neville, J. and Jensen, D., 2000. Iterative classification in relational data. AAAI.

Iterative Classification Algorithm (ICA)

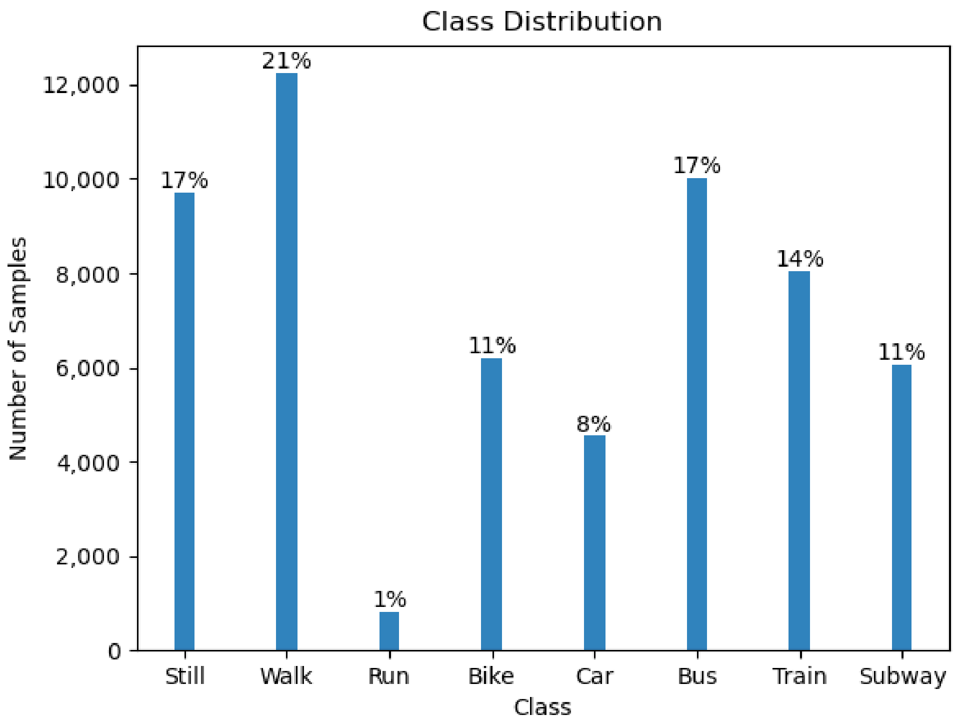
- Train classifier using the labelled instances (SVM, Random Forest, etc.)
- Until convergence
  - Apply classifier to the unlabelled nodes
  - Update the feature vectors for unlabelled nodes
- Return the labels for the labelled nodes



Neville, J. and Jensen, D., 2000. Iterative classification in relational data. AAAI.

Extension to multi-labels

- Each node has a distribution over the labels
- To avoid noise keep only the top-k labels for each unlabelled node sorted in descending order.
  - Intuition: remove the less confident labels

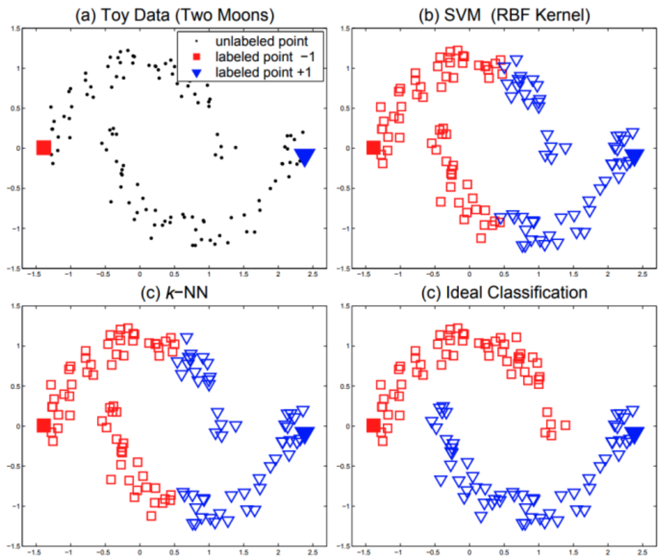


Learning with Local and Global Consistency

- The keynote of method is to let every point iteratively spread its label information to its neighbours until a global stable state is achieved.
- Input:
  - Given a point set $X = \{x_1, \ldots, x_l, x_{l+1}, \ldots, x_n\} \in R^m$
  - A label set L={1,…,c}, the first l points x_i are labelled as $y_i \in L$ and the remaining points x_u $(l+1 \le u \le n)$ are unlabelled.
- Output:
  - Predict the label of the unlabelled points

- Let F denote the set of $n \times c$ matrices with nonnegative entries. A matrix $F = [F_1, F_2, \ldots, F_n]$ corresponds to a classification on the set X by labelling each point x_i as a label $y_i = argmax_{\{j \le c\}} F_{ij}$.
- We can understand F as a vectorial function $F: X \to R^c$ which assigns a vector F_i to each point x_i.

Learning with Local and Global Consistency



The algorithm is as follows:

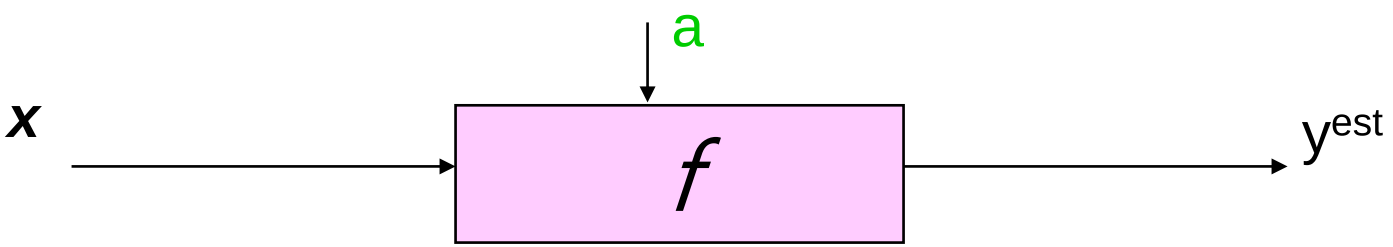
1. Form the affinity matrix W defined by:

$$W_{ij} = \begin{cases} e^{\frac{-\|x_i - x_j\|^2}{2\sigma^2}} & , \text{ if } i \neq j \\ 0 & , \text{ if } i = j \end{cases}$$

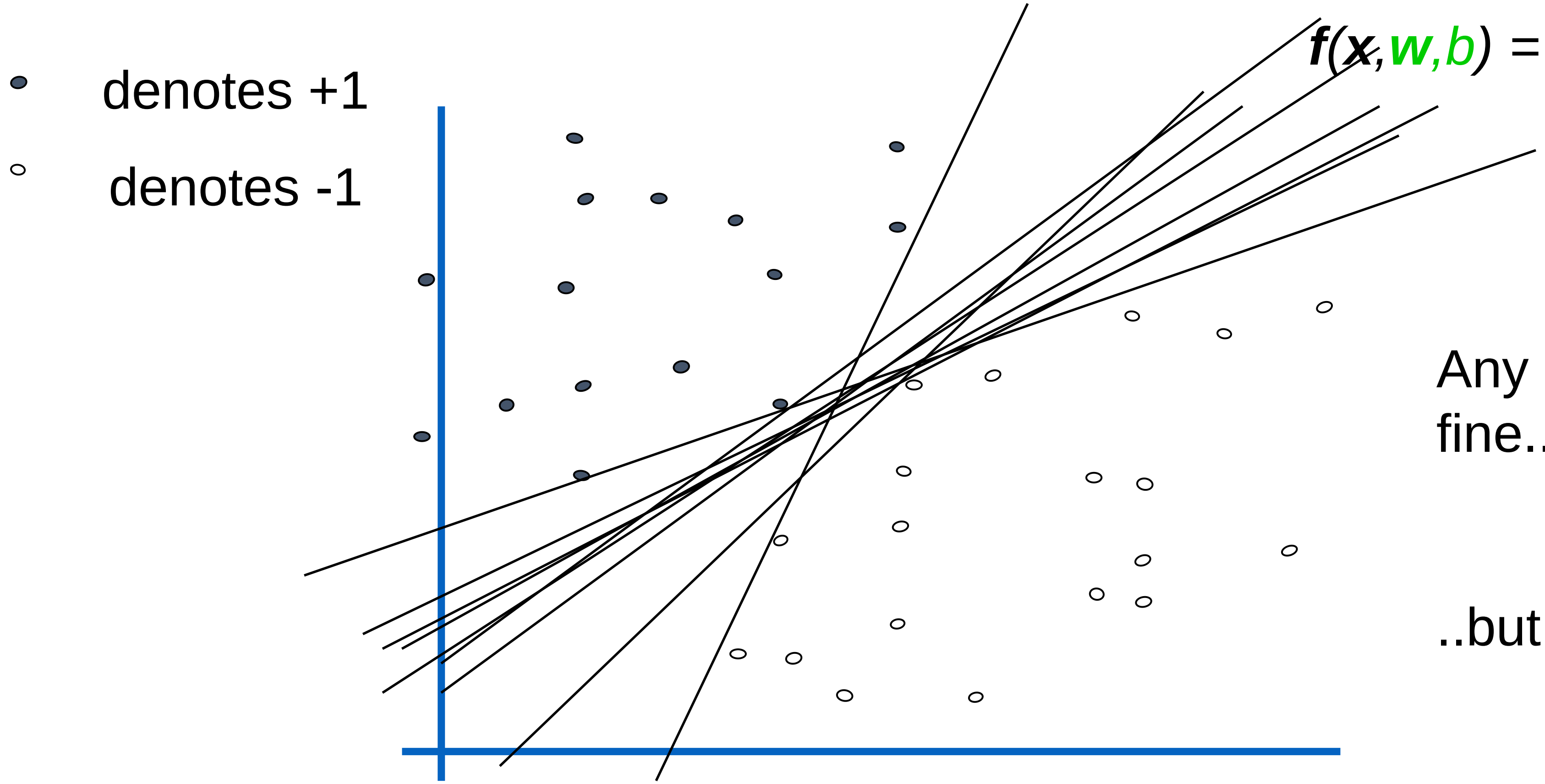
2. Construct the matrix $S = D^{-1/2} W D^{-1/2}$
   where D is a diagonal matrix with its *(i,i)*- element equal to the sum of the *i*-th row of W.

3. Iterate $F(t+1) = \alpha S F(t) + (1-\alpha) Y$ until convergence, where $\alpha \in (0,1)$

3. Finally, the label of each unlabelled point is set to be the class of which it has received most information during the iteration process.

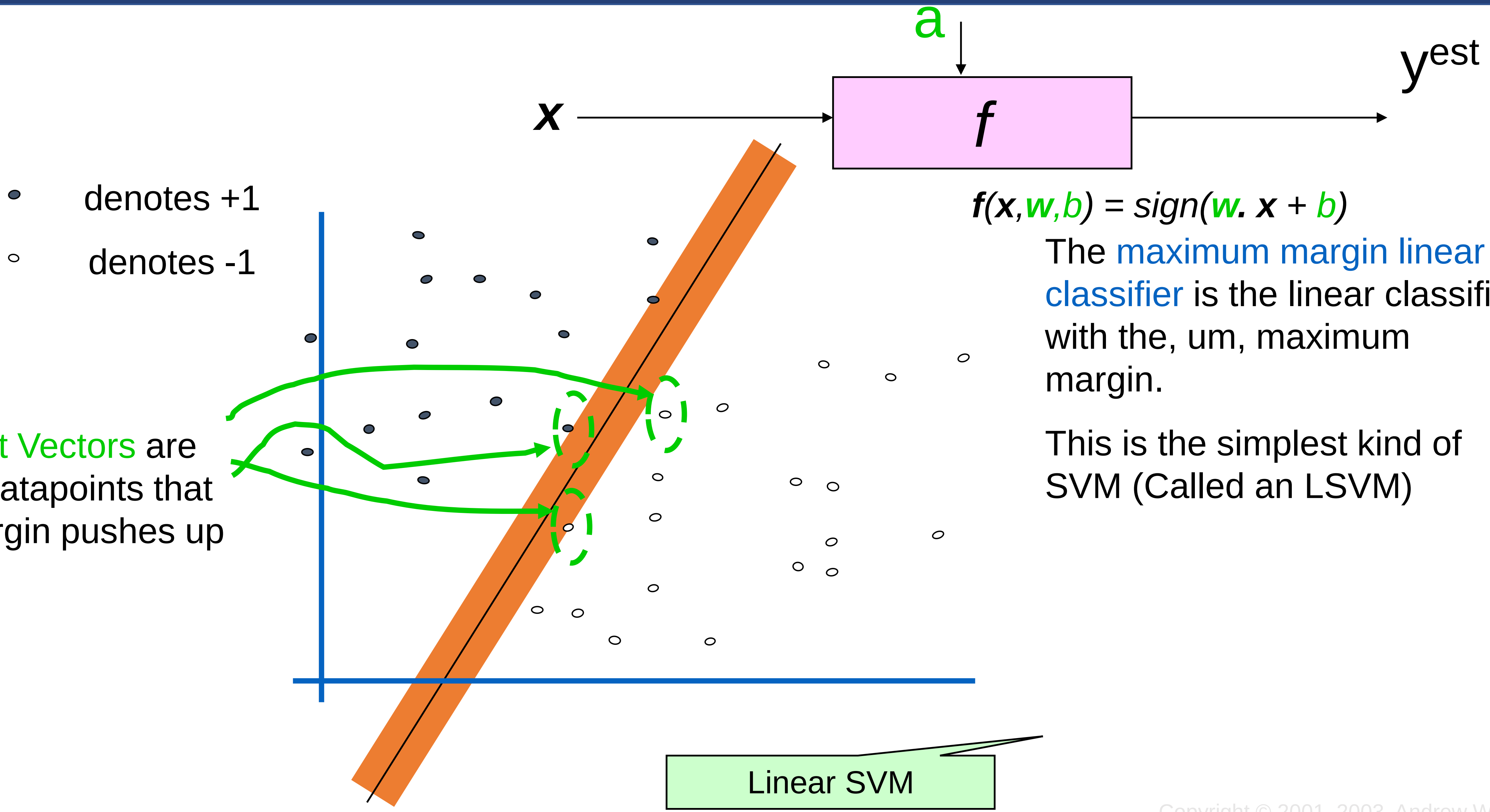


$\mathbf{f}(\boldsymbol{x},\mathbf{w},b) = sign(\mathbf{w}.\ \boldsymbol{x} + b)$



Any of these would be fine..

..but which is best?



Support Vectors are those datapoints that the margin pushes up against

$f(x,w,b) = sign(w.\ x + b)$

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Learning with Local and Global Consistency with SVM: Sample code

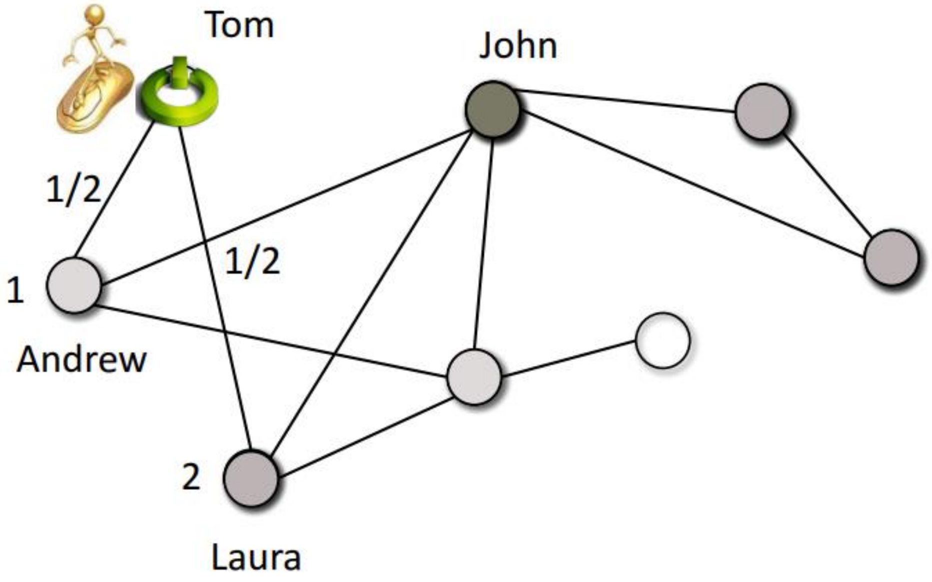

```
from networkx.algorithms import node_classification
G = nx.path_graph(4)
G.nodes[0]['label'] = 'A'
G.nodes[3]['label'] = 'B'
G.nodes(data=True)

G.edges()

predicted = node_classification.local_and_global_consistency(G)
predicted
```

```
['A', 'A', 'B', 'B']
```

Label propagation

- Random Walk
- Personalized Random Walk with Restarts (RWR)
- PageRank: A kind of random walk
- Set of neighbours of nodes

- In a random walk, you assume that the walker moves randomly and chooses one of the neighbours to visit.
- In the figure:
  - A chooses B or C with probability 1/2.
  - Once he chooses one it increases the number of times he visited that node
  - Continue the process until nothing changes anymore (at a probabilistic level)



D will receive many visits since many nodes are connected to him

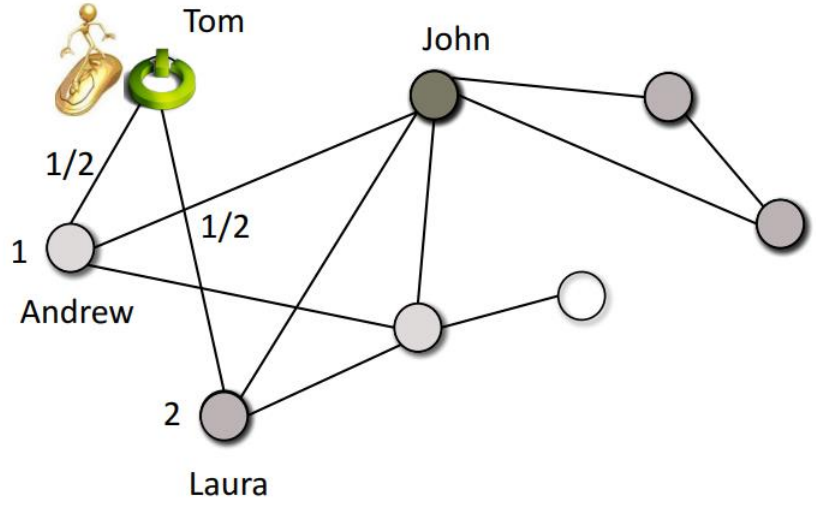
- Now assume that with probability c you perform another move and with probability (1-c) you jump back to Tom.
- Therefore the probability for the walker of being in Tom place will be:

Probability of visiting Andrew at time (t-1)

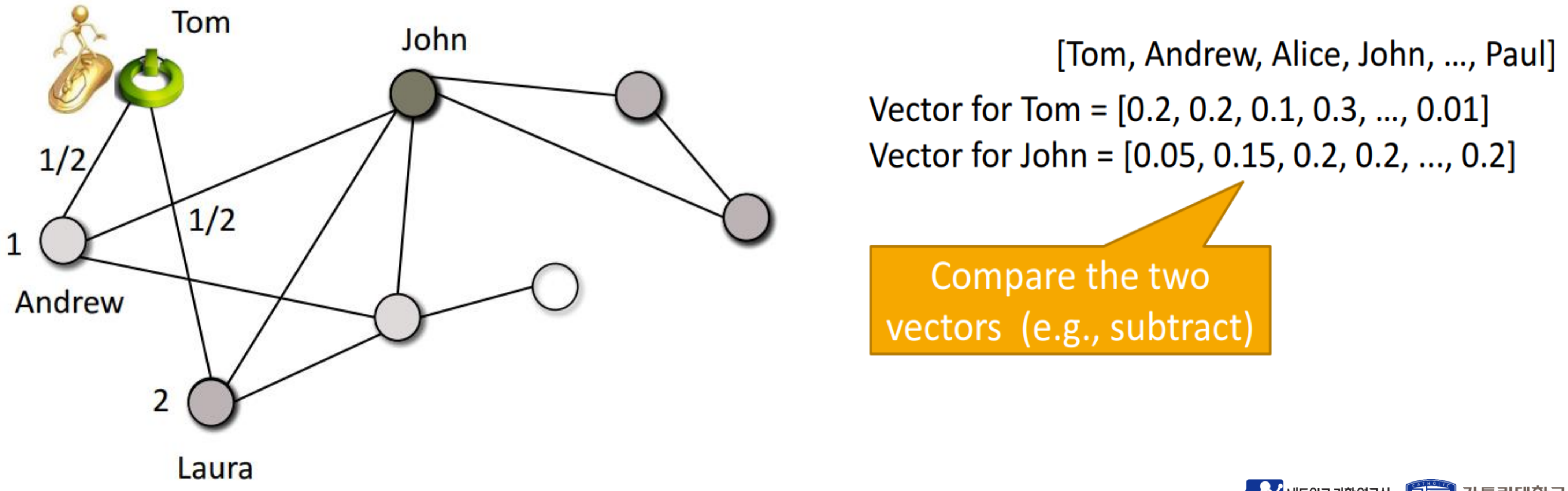
$$Tom(t) = c\,\frac{Prob(Andrew)(t-1) + Prob(Laura)(t-1)}{3} + (1-\mathrm{c})$$

Number of Andrew and Laura's neighbors

Probability of jumping back to Tom



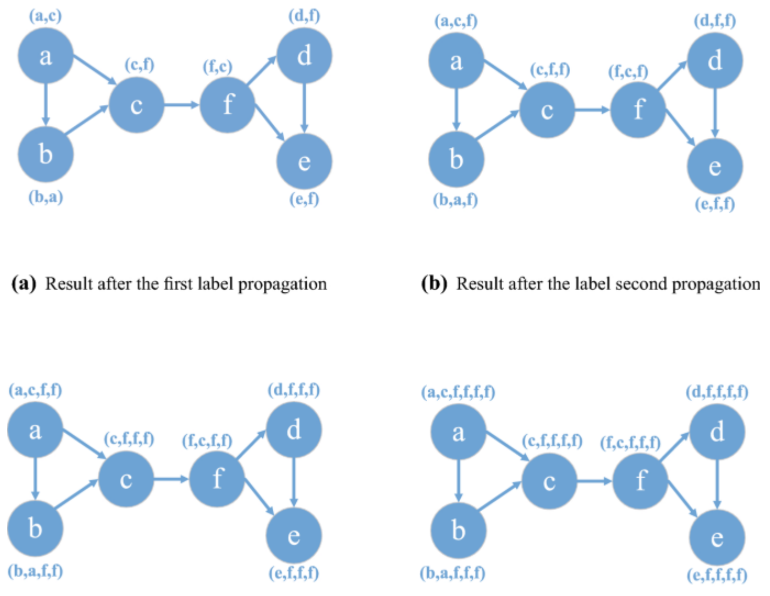
- Start two random walks from the two nodes you want to compare separately
- Compare the final scores you obtain for each node in the graph using some vector comparison (e.g., cosine similarity, KL-divergence)



Labels of neighbours of target nodes

- At each step of the process, each vertex updates its label to a new one which corresponds to the most frequent label among its neighbours.
- For each vertex $v \in V$, v updates its label according to:

$$l_v = \arg\max \sum_{u \in N(v)} [l_u == l]$$

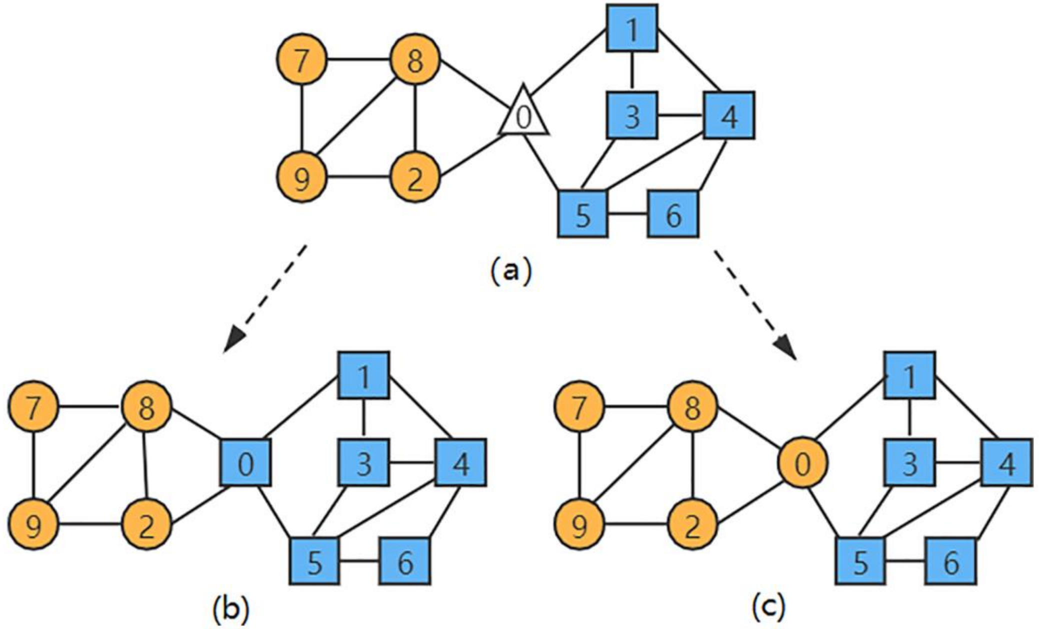


(a) Result after the first label propagation

(b) Result after the label second propagation

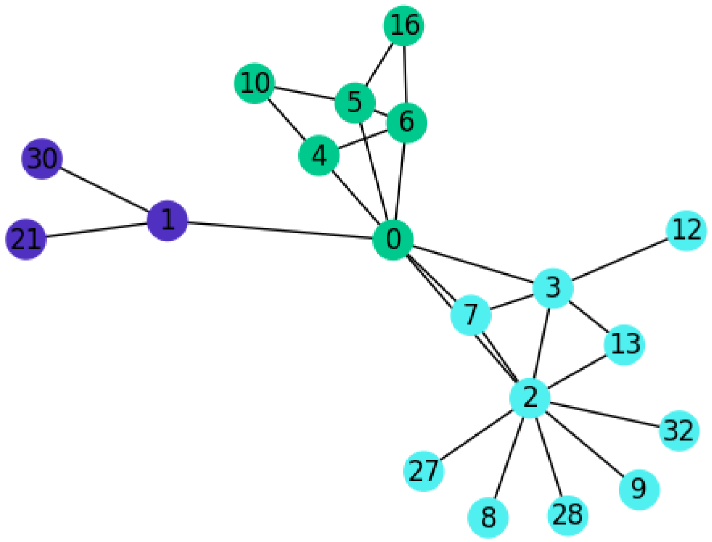
(c) Result after the third label propagation

(d) Result after the fourth label propagation



Label propagation: Sample code

```
colors = ["#00C98D", "#5030C0", "#50F0F0"]
pos = nx.spring_layout(G)
lst_m = community.label_propagation_communities(G)
color_map_b = {}
keys = G.nodes()
values = "black"
for i in keys:
        color_map_b[i] = values
counter = 0
for c in lst_m:
  for n in c:
    color_map_b[n] = colors[counter]
  counter = counter + 1
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_nodes(G, pos, node_color=dict(color_map_b).values())
nx.draw_networkx_labels(G, pos)
plt.axis("off")
plt.show()
```



Evaluation metrics

- Precision/Recall
- Accuracy + weighted loss
- ROC and AUC

Precision and Recall

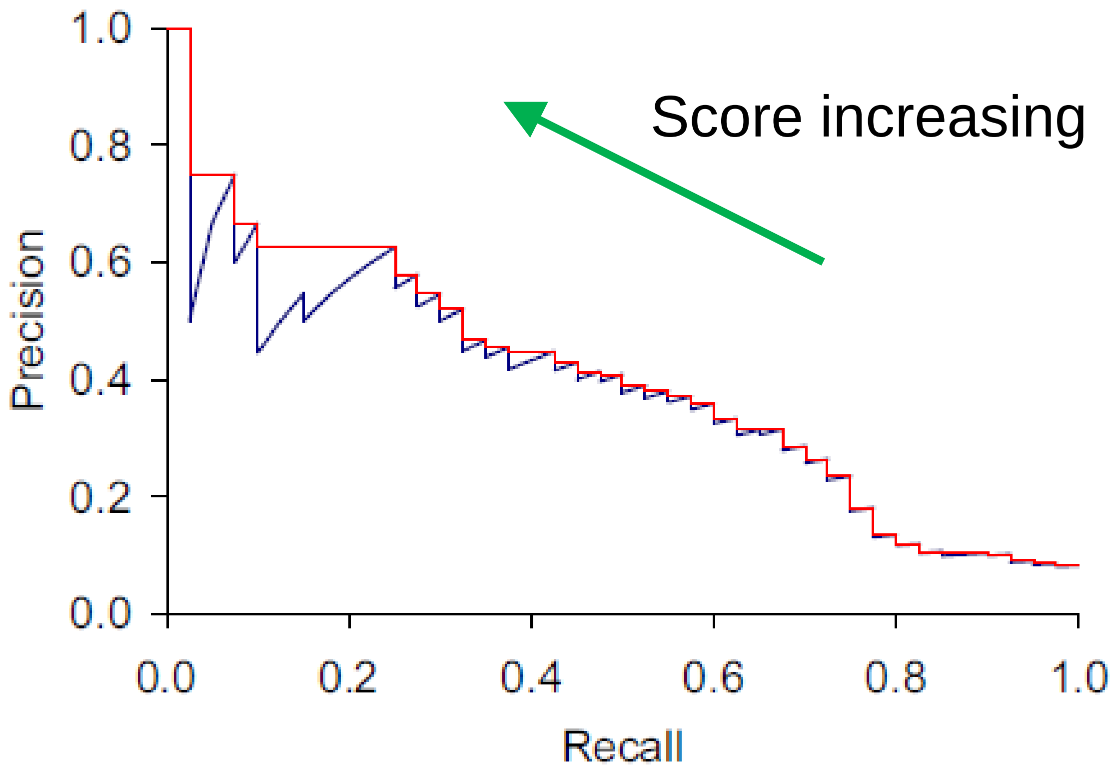
- When evaluating a search tool or a classifier, we are interested in at least two performance measures:

- **Precision:** Within a given set of positively-labeled results, the fraction that were true positives = tp/(tp + fp)

- **Recall:** Given a set of positively-labeled results, the fraction of all positives that were retrieved = tp/(tp + fn)

- Positively-labeled means judged “relevant” by the search engine or labeled as in the class by a classifier. tp = true positive, fp = false positive etc.

- Search tools and classifiers normally assign scores to items. Sorting by score gives us a precision-recall plot which shows what performance would be for different score thresholds.



Be careful of "Accuracy"

- The simplest measure of performance would be the fraction of items that are correctly classified, or the "accuracy" which is:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

- But this measure is dominated by the larger set (of positives or negatives) and favours trivial classifiers.

- e.g. if 5% of items are truly positive, then a classifier that always says "negative" is 95% accurate.

Weighted loss

We can instead try to minimize a weight sum:

$$w_1 \text{ fn} + w_2 \text{ fp}$$

And typically, $w_1 \gg w_2$, since positives are often much rarer (clicks or purchases or viewing a movie).

The weighted "F" measure

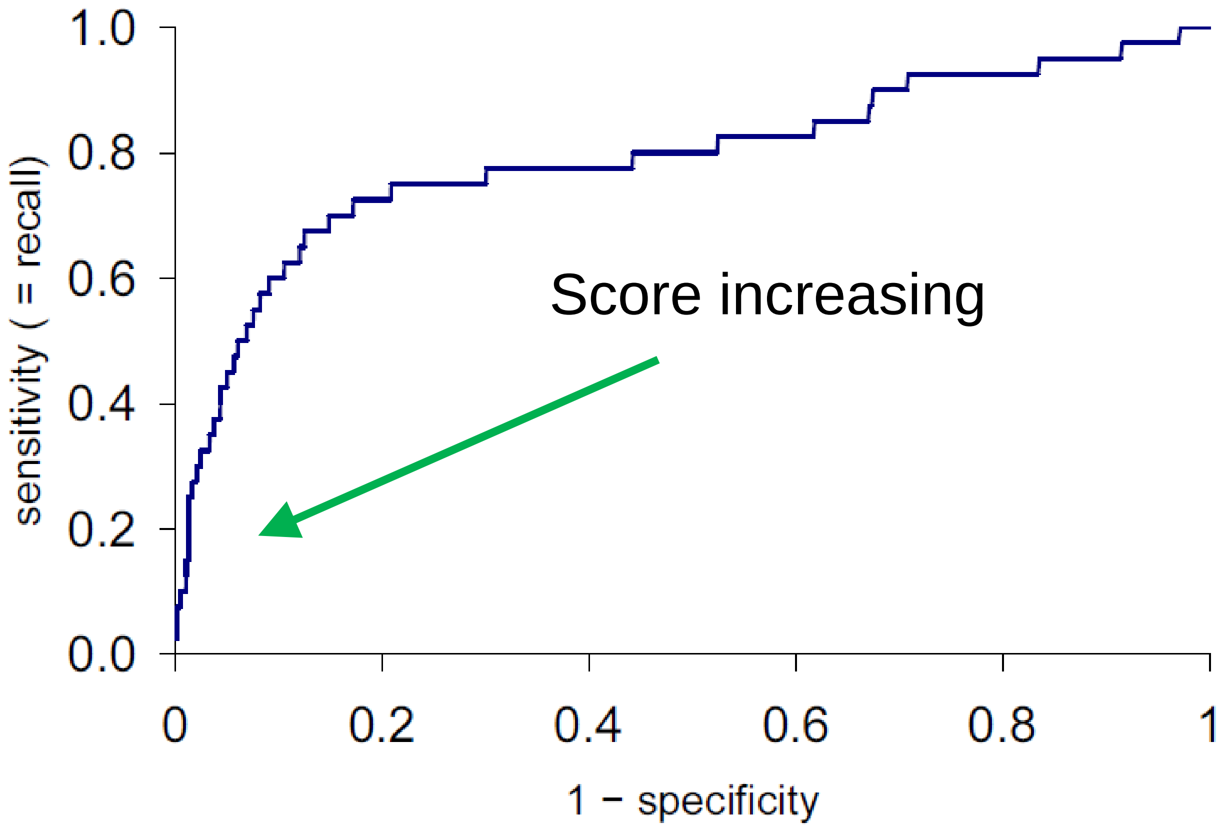
A measure that naturally combines precision and recall is the β-weighted F-meas ure:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

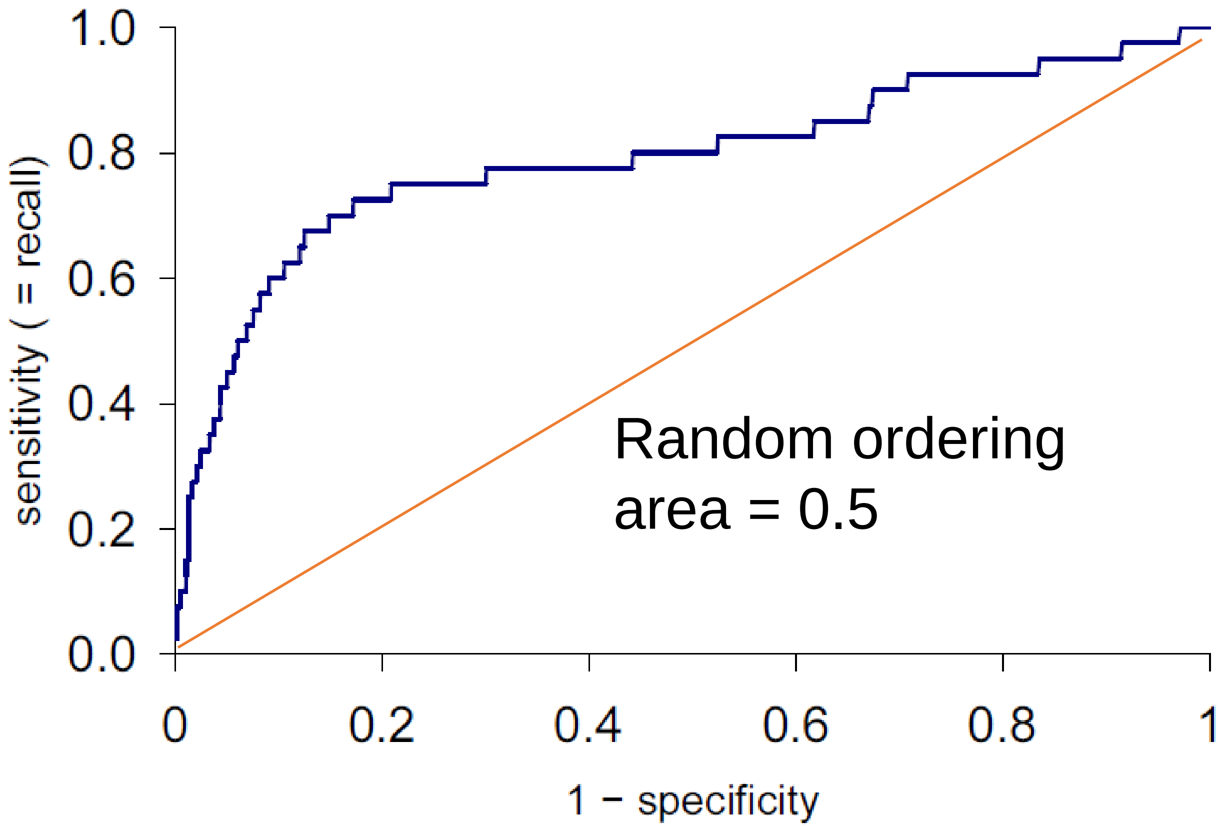
Which is the weighted harmonic mean of precision and recall. Setting β = 1 gives us the F_1 – measure. It can also be computed as:

$$F_{\beta=1} = \frac{2PR}{P + R}$$

ROC is Receiver-Operating Characteristic. ROC plots
Y-axis: true positive rate = tp/(tp + fn), same as recall
X-axis: false positive rate = fp/(fp + tn) = 1 - specificity



- ROC AUC is the "Area Under the Curve" – a single number that captures the overall quality of the classifier. It should be between 0.5 (random classifier) and 1.0 (perfect).



Q A

네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA