Applications

Node Classification

Acknowledgement: Jure Leskovec, Stanford University

Overview

- Node Classification
- Relational Classifiers
- Iterative Classification
- Belief Propagation

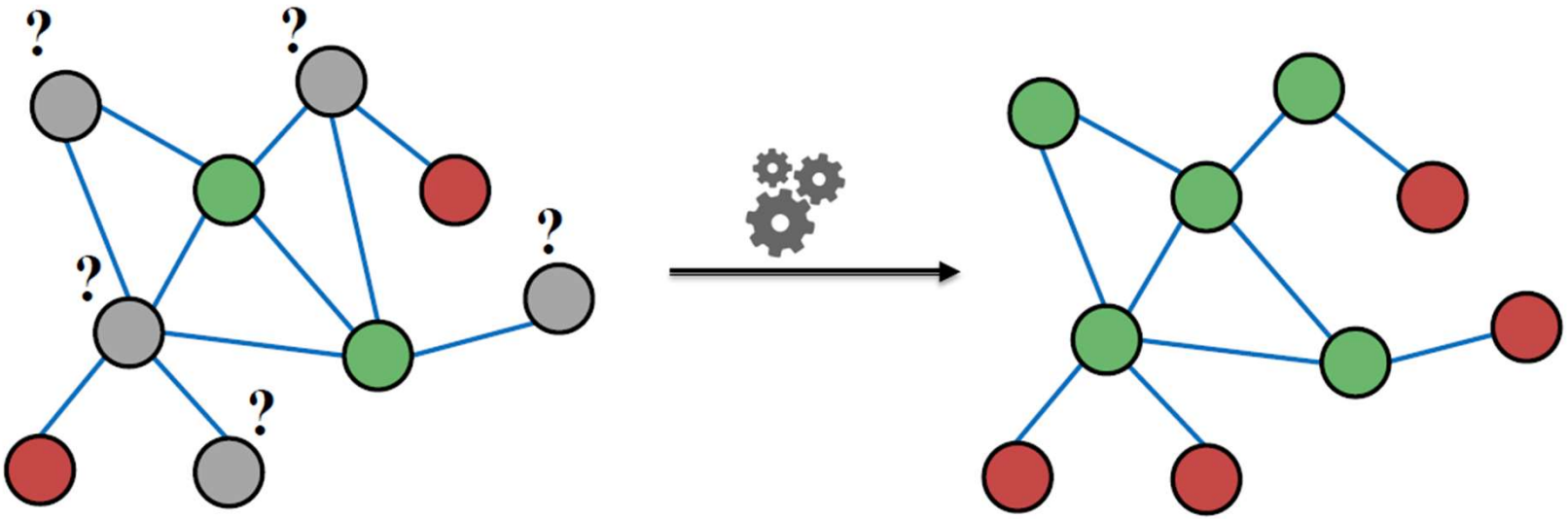
Overview

- Node Classification
- Relational Classifiers
- Iterative Classification
- Belief Propagation

Node Classification

- Given a network with labels on some nodes, how do we assign labels to all other nodes in the network?
- Example:
  - In a network, some nodes are fraudsters and some other nodes are fully trusted.
  - How do you find the other fraudsters and trustworthy nodes?
- Can utilize a variety of techniques
  - Machine Learning
  - SNA

Node Classification



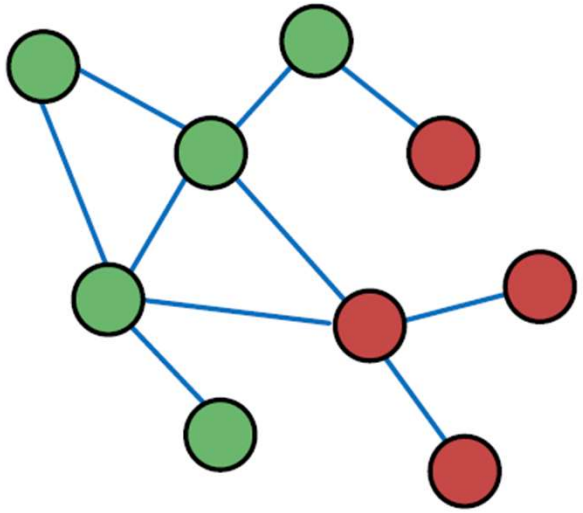
- Given labels of some nodes
- Let's predict labels of unlabeled nodes
- This is called semi-supervised node classification

Node Classification

- **A framework for Node Classification: Message Passing**
- **Intuition: Correlations** exist in networks.
  - In other words: Similar nodes are connected
- **Key concept** is **collective classification**: Idea of assigning labels to all nodes in a network together
- **We will look at three techniques:**
  1. **Relational classification**
  2. **Iterative classification**
  3. **Belief propagation**

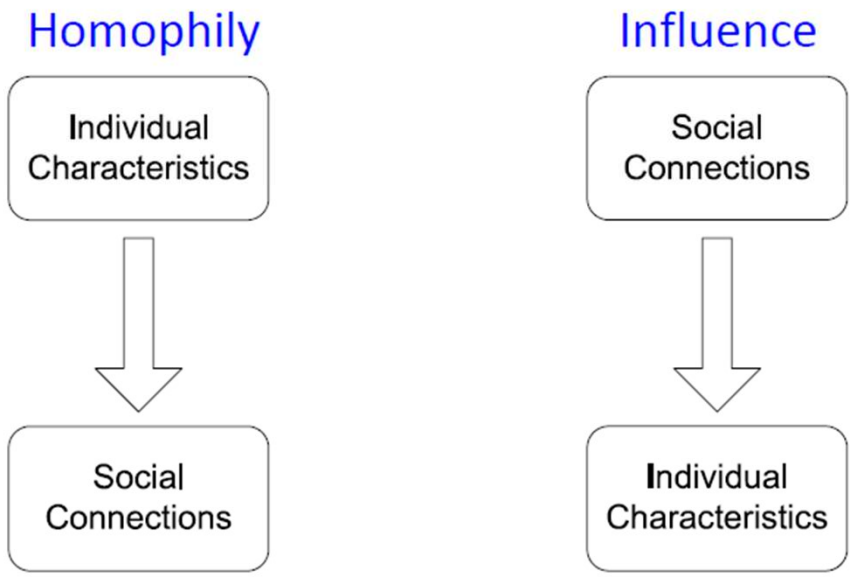
Correlations exist in the Network

- Individual behaviors are **correlated** in the network
- **Correlation**: nearby nodes have the same color (belonging to the same class)



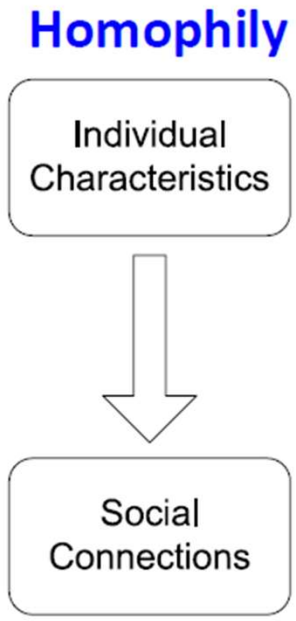
Correlations exist in the Network

- **Main types of dependencies that lead to correlation:**



Homophily

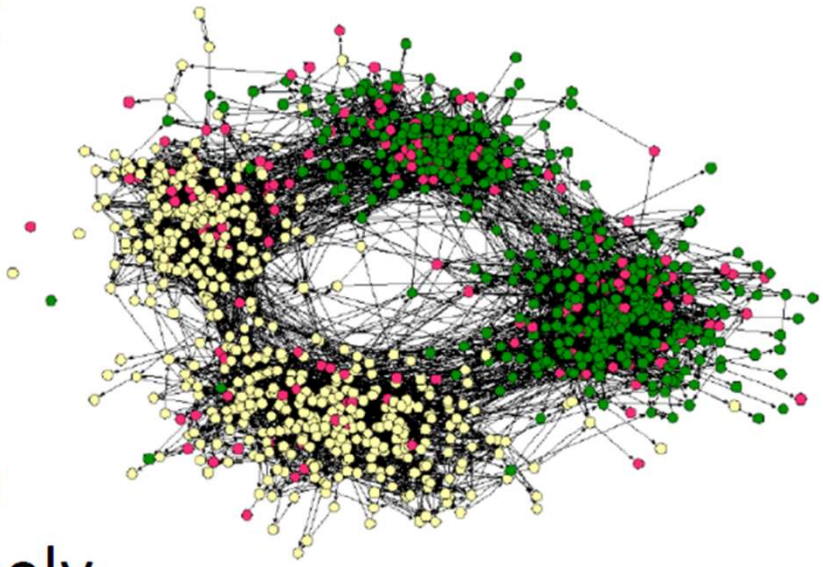
- **Homophily**: The tendency of individuals to associate and bond with similar others
  - *"Birds of a feather flock together"*
  - It has been observed in a vast array of network studies, based on a variety of attributes (e.g., age, gender, organizational role, etc.)
  - **Example**: Researchers who focus on the same research area are more likely to establish a connection (meeting at conferences, interacting in academic talks, etc.)



Homophily: Example

Example of homophily

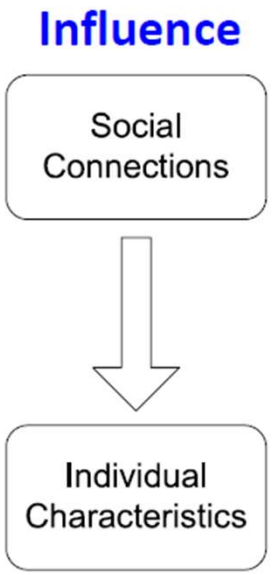
- Online social network
  - Nodes = people
  - Edges = friendship
  - Node color = interests (sports, arts, etc.)
- People with the same interest are more closely connected due to homophily



(Easley and Kleinberg, 2010)

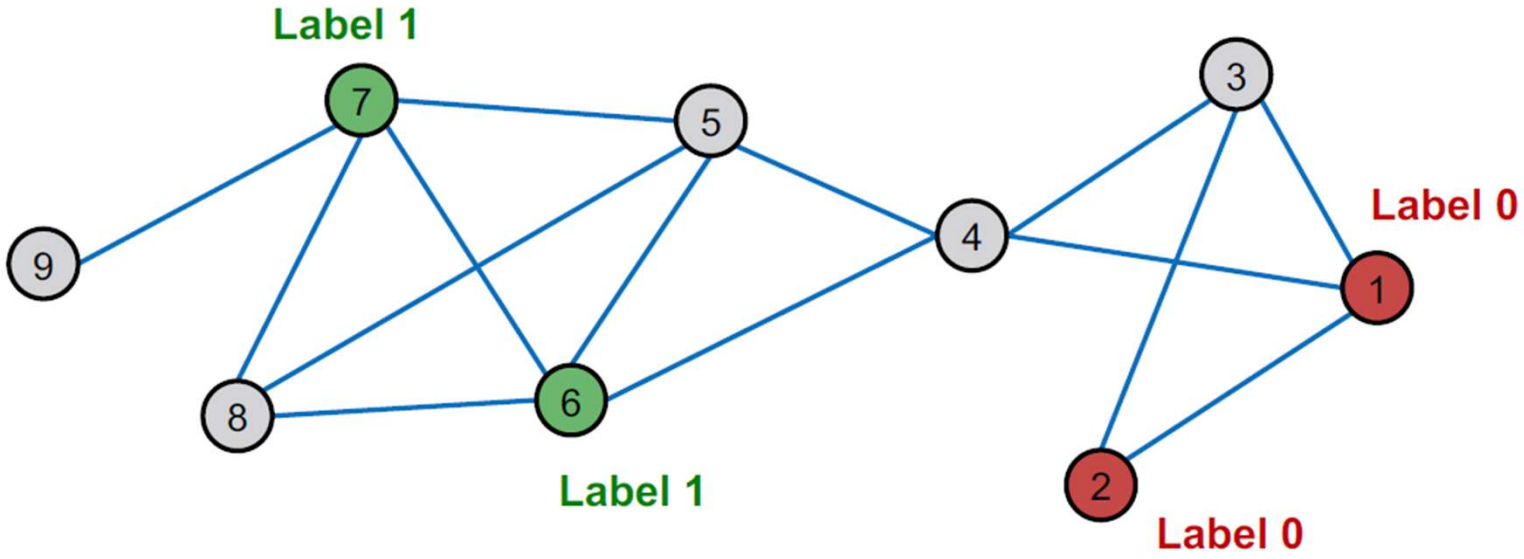
Influence

- **Influence**: Social connections can influence the individual characteristics of a person.
  - **Example**: I recommend my musical preferences to my friends, until one of them grows to like my same favorite genres!



Classification with Network Data

- How do we leverage this correlation observed in networks to help predict node labels?



How do we predict the labels for the nodes in grey?

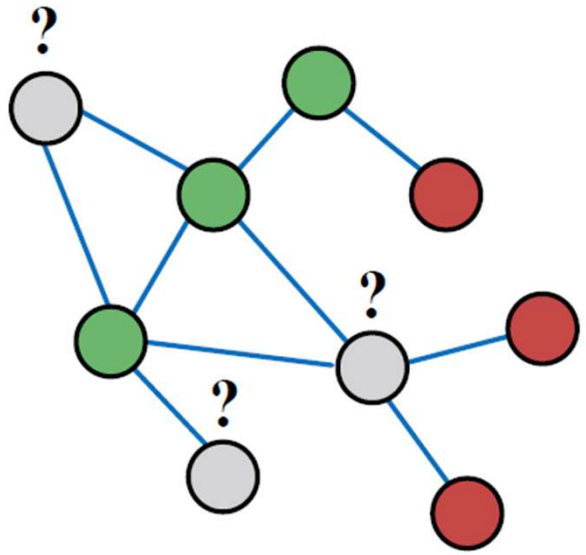
Motivation

- **Similar nodes are typically close together or directly connected in the network:**
  - **Guilt-by-association**: If I am connected to a node with label X, then I am likely to have label X as well.
  - **Example: Malicious/benign web page:** Malicious web pages link to one another to increase visibility, look credible, and rank higher in search engines

Motivation

- **Classification label** of a node v in network may depend on:
  - Features of v
  - Labels of the nodes in v's neighborhood
  - Features of the nodes in v's neighborhood

Semi-supervised Learning



Given:
- Graph
- Few labeled nodes

Find: class (red/green) of remaining nodes

Main assumption: There is homophily in the network

Semi-supervised Learning

Example task:

- Let A be a $n \times n$ adjacency matrix over n nodes
- Let $\mathrm{Y} = \{0, 1\}^n$ be a vector of **labels**:
  - $\mathrm{Y}_v = 1$ belongs to Class 1
  - $\mathrm{Y}_v = 0$ belongs to Class 0
  - There are unlabeled node needs to be classified
- **Goal:** Predict which **unlabeled** nodes are likely **Class 1**, and which are likely **Class 0**

Approach: Collective Classification

- **Many applications:**
  - Document classification
  - Part of speech tagging
  - Link prediction
  - Optical character recognition
  - Image/3D data segmentation
  - Entity resolution in sensor networks
  - Spam and fraud detection

Collective Classification Overview

- **Intuition**: Simultaneous classification of interlinked nodes using correlations
- Probabilistic framework
- Markov Assumption: *the label Y_v of one node v depends on the labels of its neighbors N_v*

$$P(Y_v) = P(Y_v|N_v)$$

- Collective classification involves 3 steps:

Local Classifier	Relational Classifier	Collective Inference
• Assign initial labels	• Capture correlations between nodes	• Propagate correlations through network

Collective Classification Overview

Local Classifier

- Assign initial labels

Relational Classifier

- Capture correlations between nodes

Collective Inference

- Propagate correlations through network

Local Classifier: Used for initial label assignment

- Predicts label based on node attributes/features
- Standard classification task
- Does not use network information

Relational Classifier: Capture correlations

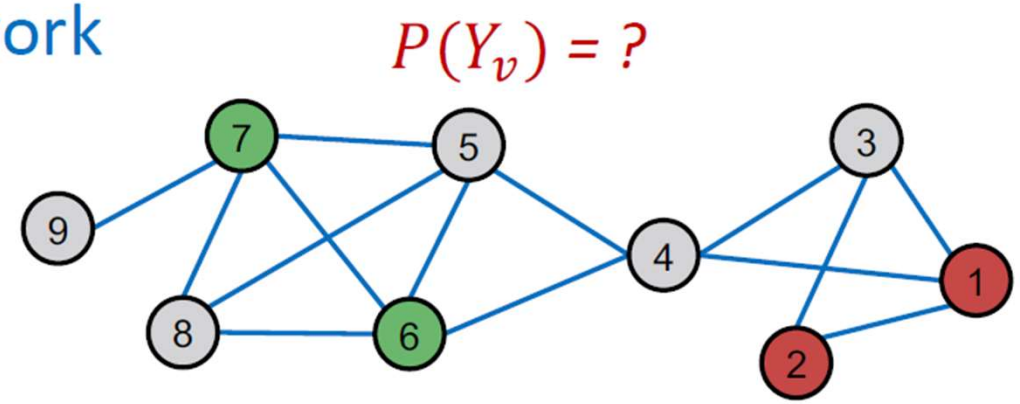
- Learns a classifier to label one node based on the labels and/or attributes of its neighbors
- This is where network information is used

Collective Inference: Propagate the correlation

- Apply relational classifier to each node iteratively
- Iterate until the inconsistency between neighboring labels is minimized
- Network structure affects the final prediction

Problem Setting

- How to predict the labels Y_v for the unlabeled nodes v (in grey color)?
- Each node v has a feature vector f_v
- Labels for some nodes are given (1 for green, 0 for red)
- **Task:** Find $P(Y_v)$ given all features and the network



Overview

- Node Classification
- Relational Classifiers
- Iterative Classification
- Belief Propagation

Probabilistic Relational Classifier

- **Basic idea:** Class probability Y_v of node v is a weighted average of class probabilities of its neighbors
- For **labeled nodes** v, initialize label Y_v with ground-truth label Y_v^*
- For **unlabeled nodes**, initialize $Y_v = 0.5$
- **Update** all nodes in a random order until convergence or until maximum number of iterations is reached

Probabilistic Relational Classifier

- **Update** for each node v and label c (e.g. 0 or 1)

$$P(Y_v = c) = \frac{1}{\sum_{(v,u)\in E} A_{v,u}} \sum_{(v,u)\in E} A_{v,u}\, P(Y_u = c)$$

  - If edges have strength/weight information, $A_{v,u}$ can be the edge weight between v and u
  - $P(Y_v = c)$ is the probability of node v having label c

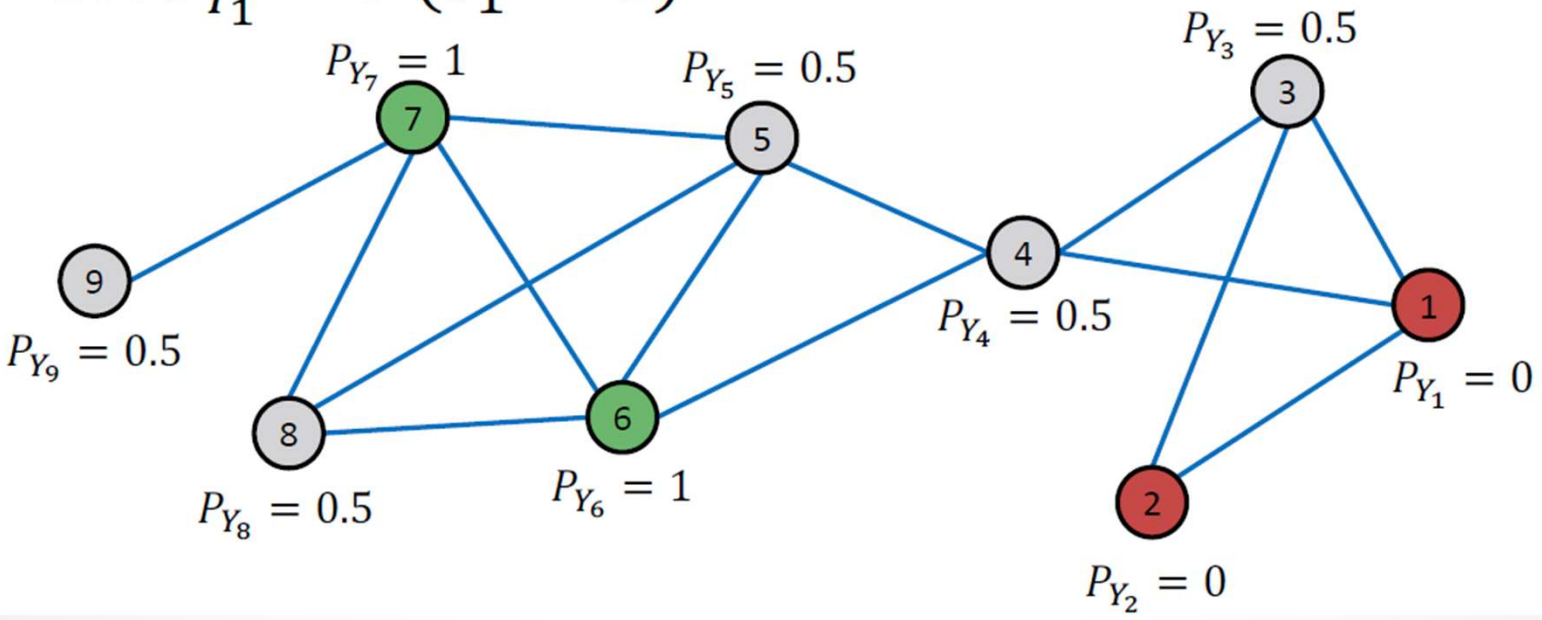
- Challenges:
  - Convergence is not guaranteed
  - Model cannot use node feature information

Example: Initiation

Initialization:

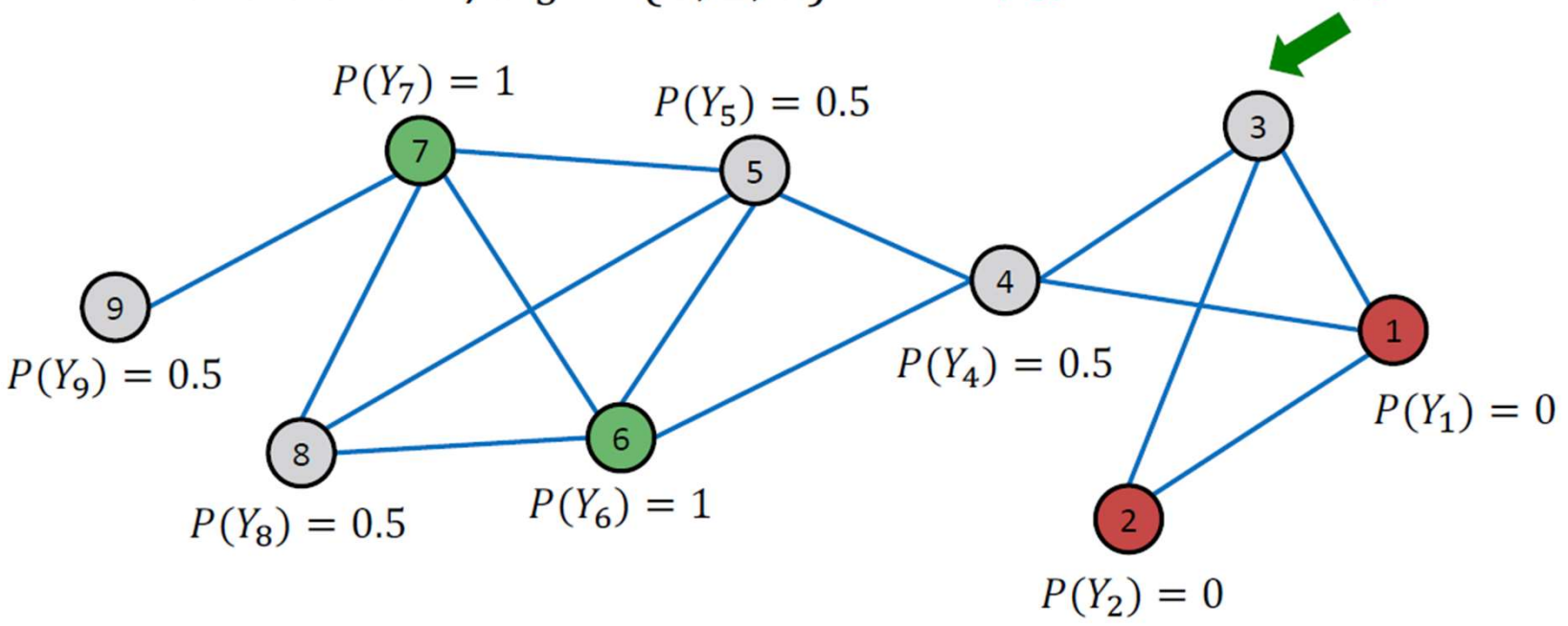
- All labeled nodes with their labels
- All unlabeled nodes 0.5 (belonging to class 1 with probability 0.5)

Let $P_{Y_1} = P(Y_1 = 1)$



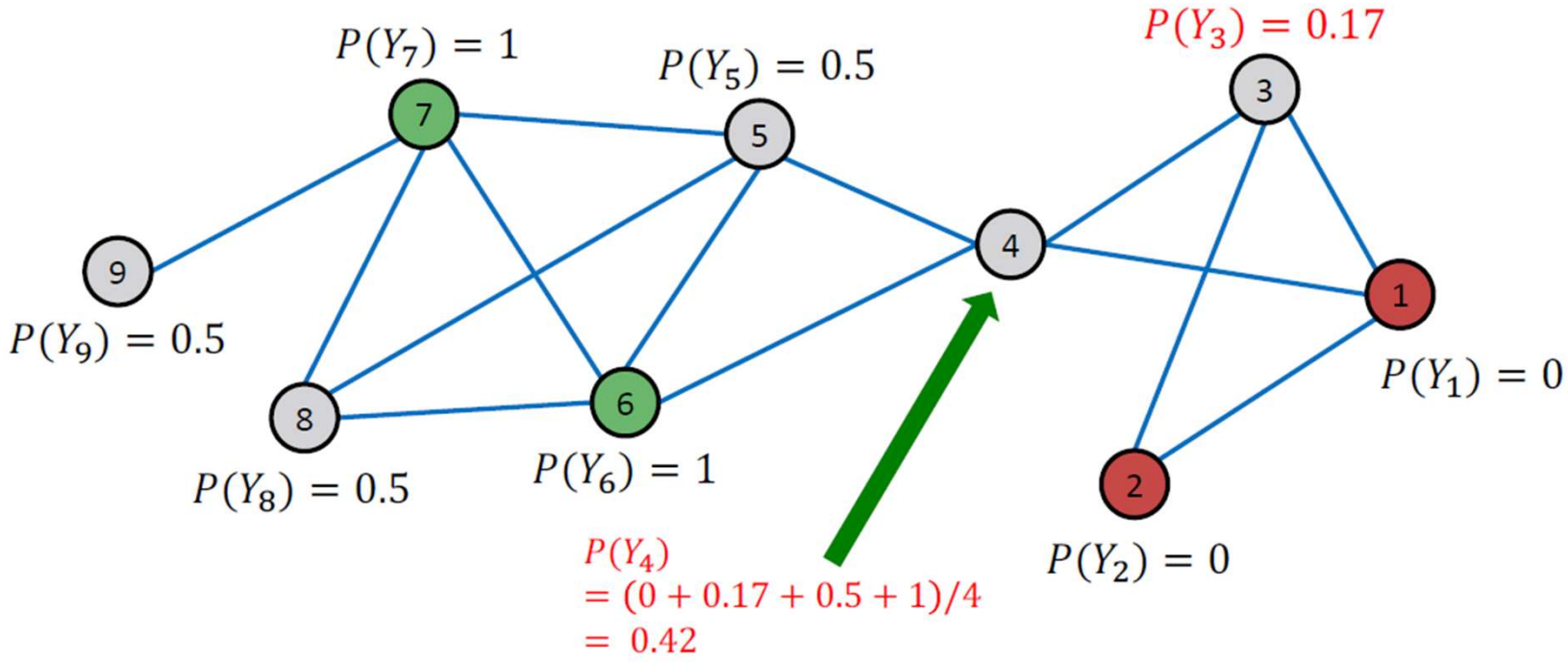
Example: 1st Iteration, Update Node 3

- Update for the 1st Iteration:
  - For node 3, $N_3 = \{1, 2, 4\}$



Example: 1st Iteration, Update Node 4

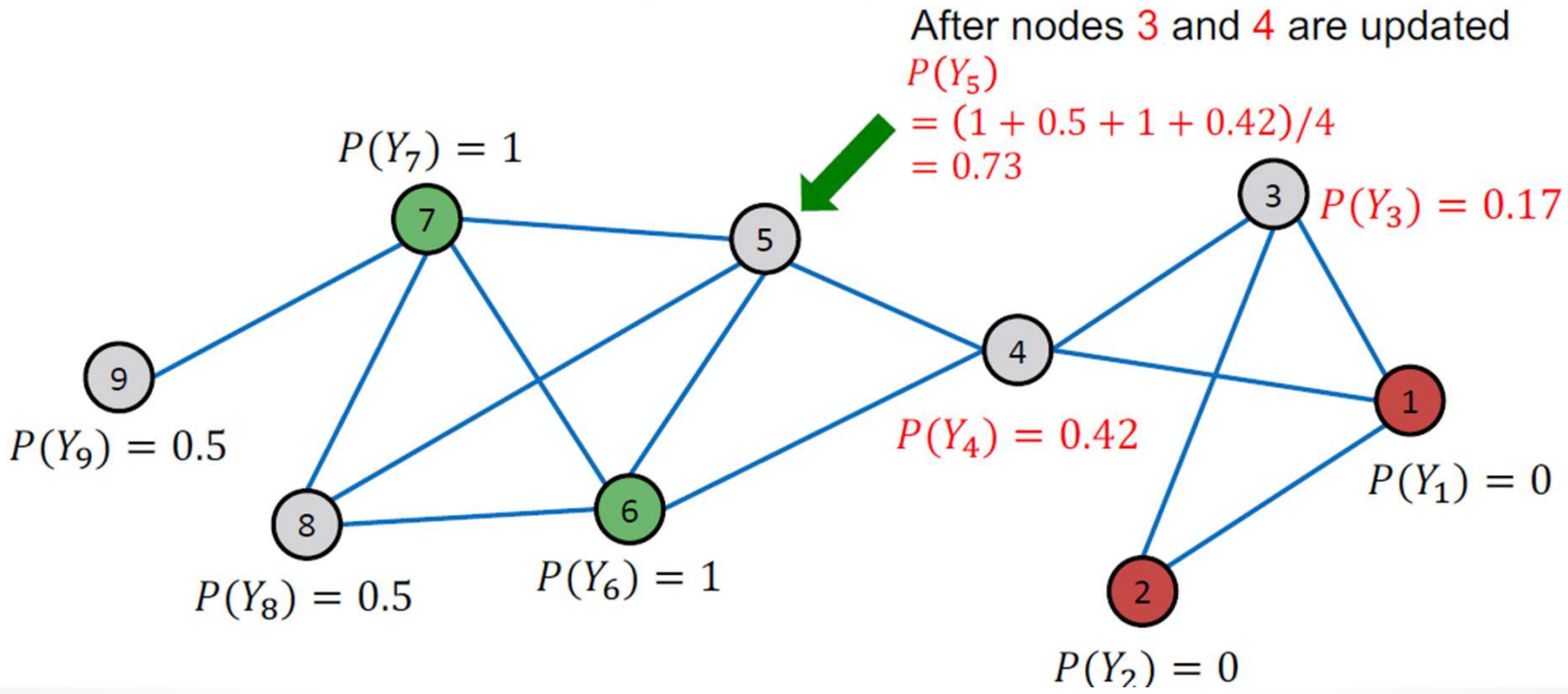
- Update for the 1st Iteration:
  - For node 4, $N_4 = \{1, 3, 5, 6\}$



After Node 3 is updated

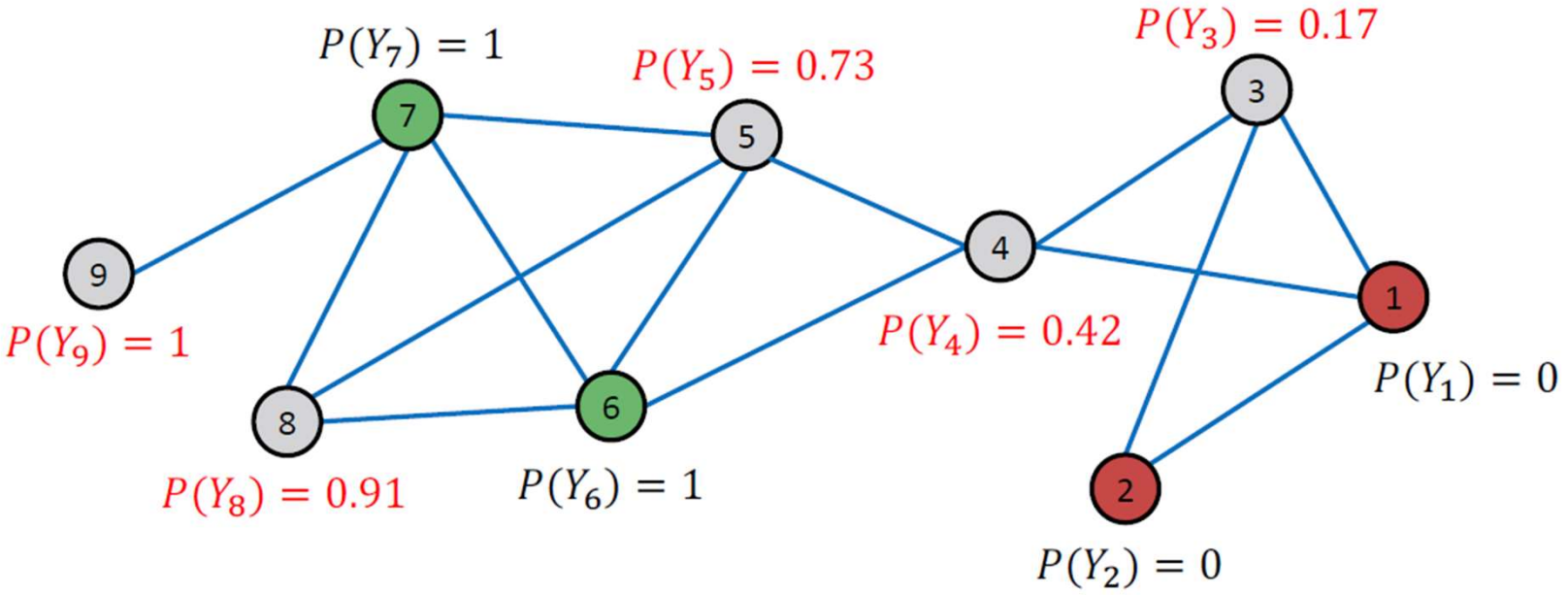
Example: 1st Iteration, Update Node 5

- Update for the 1st Iteration:
  - For node 5, $N_5 = \{4, 6, 7, 8\}$

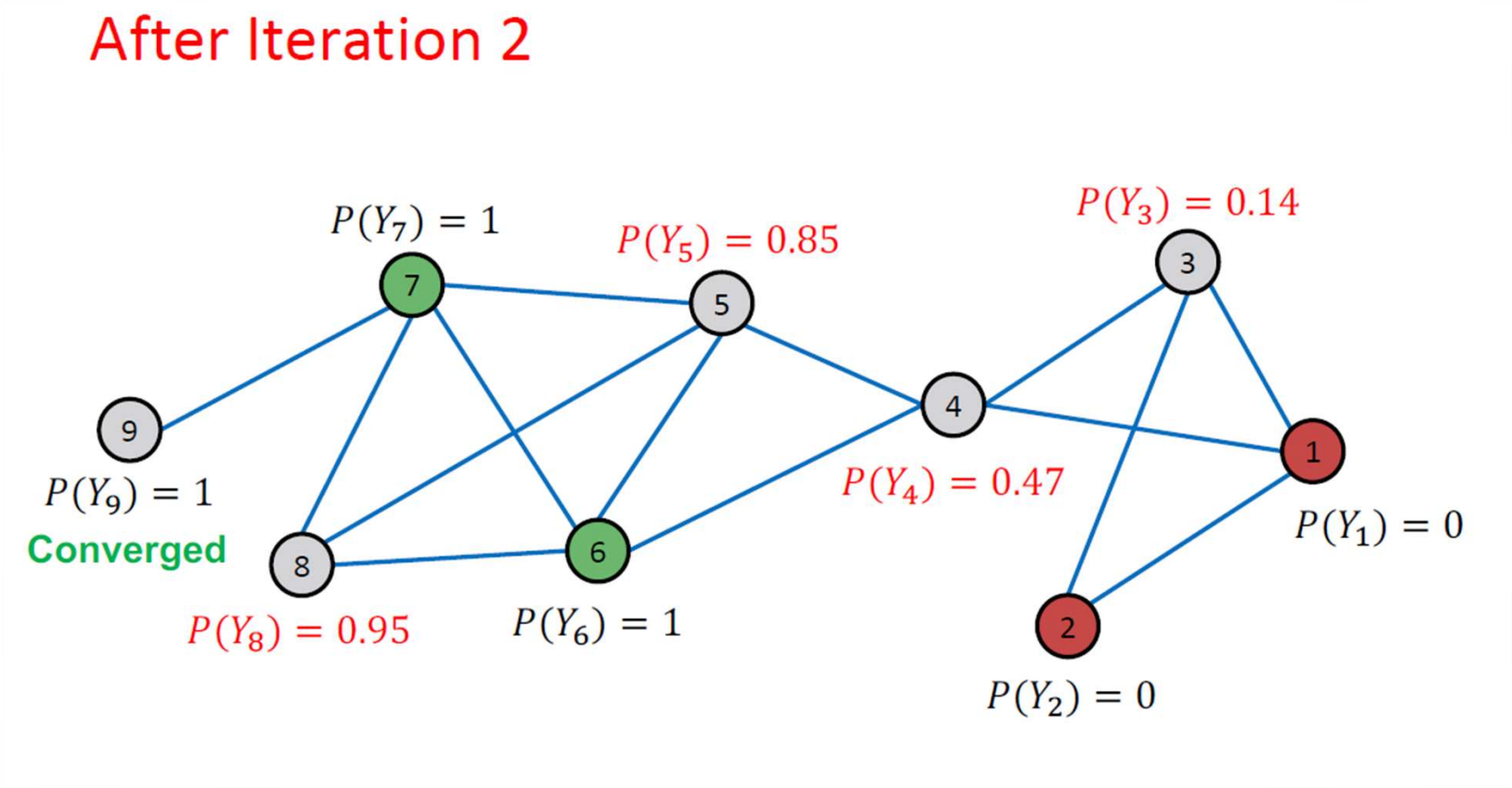


Example: After 1st Iteration

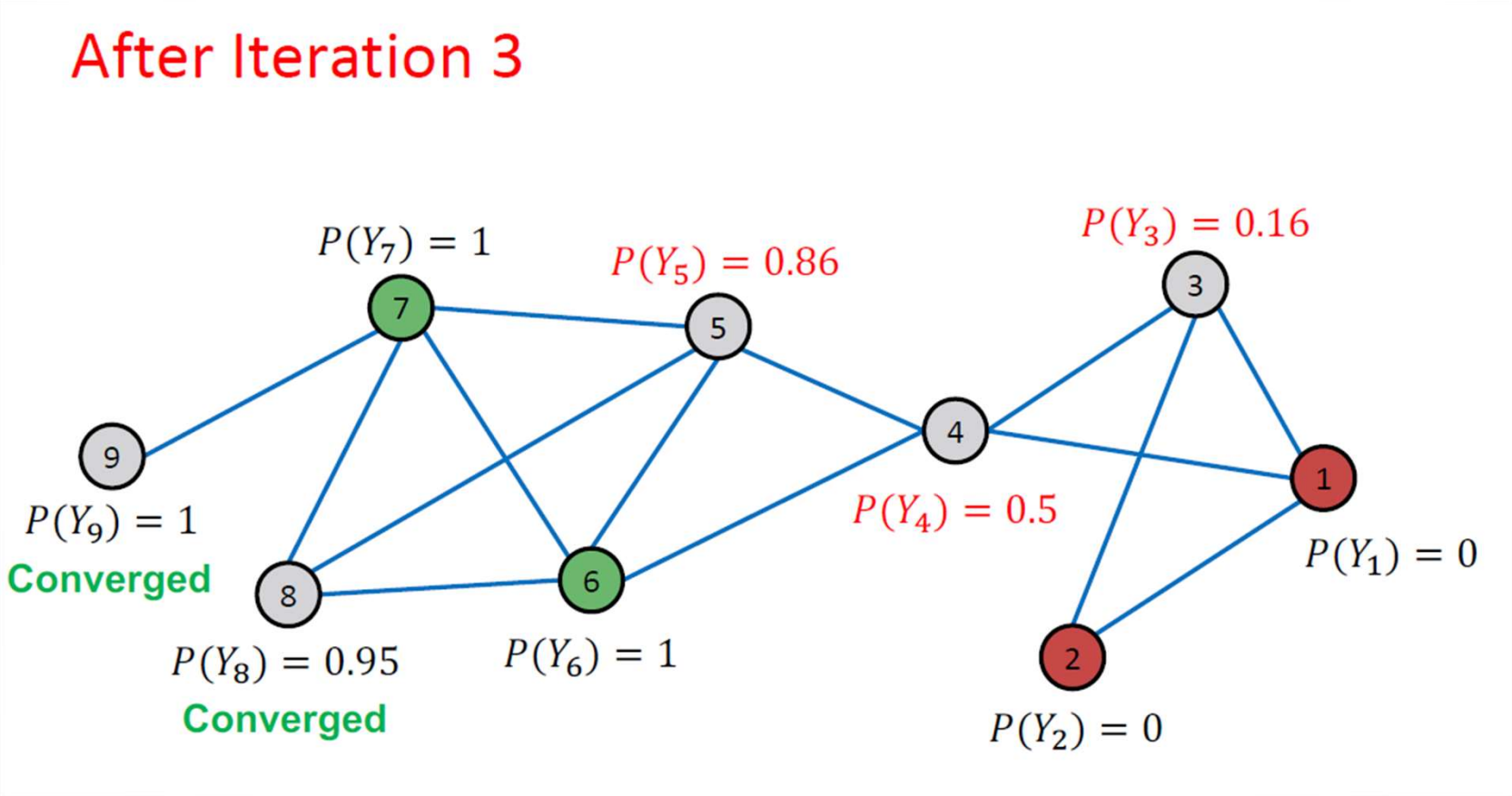
After Iteration 1 (a round of updates for all unlabeled nodes)



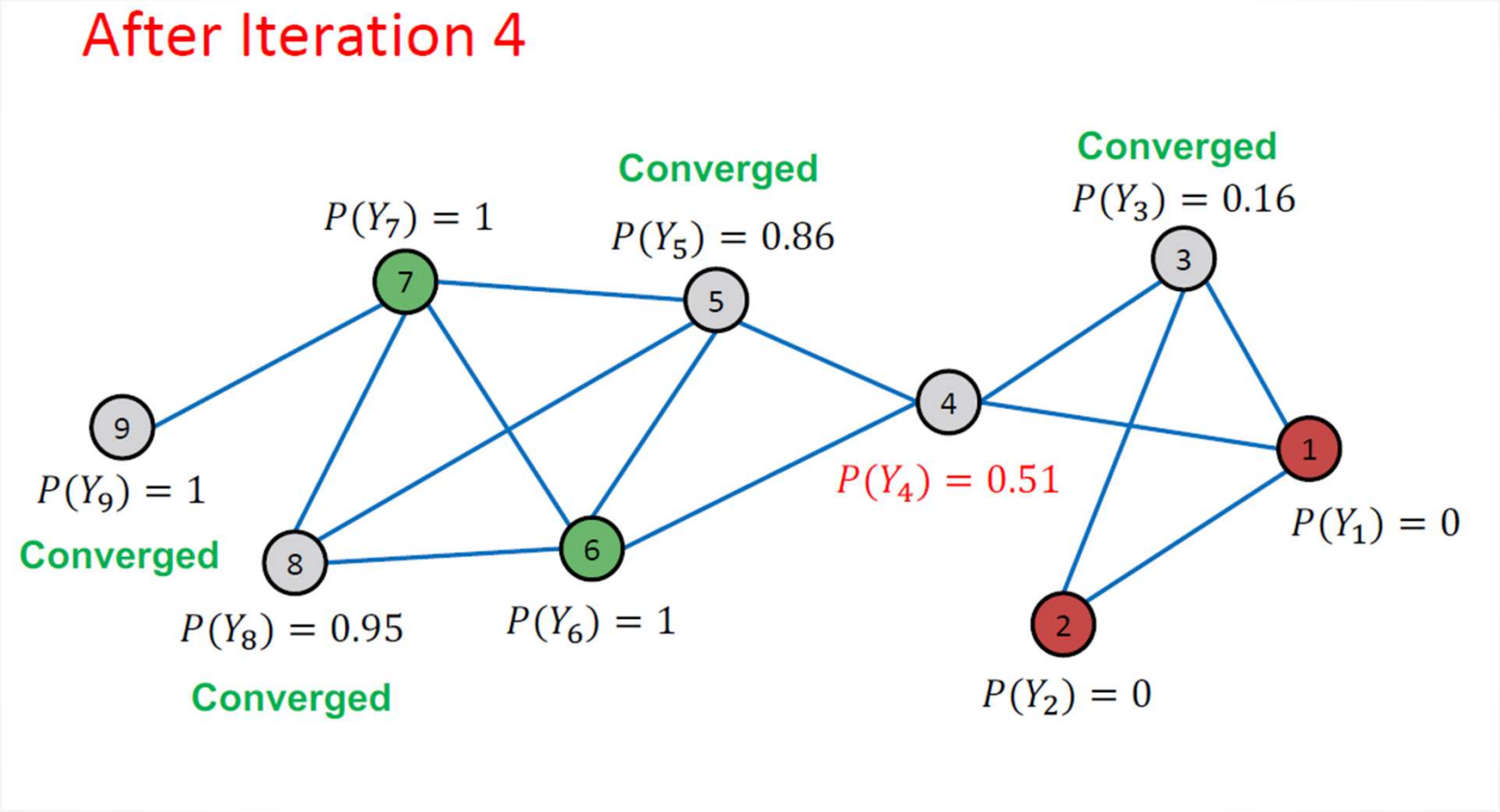
Example: After 2nd Iteration



Example: After 3rd Iteration

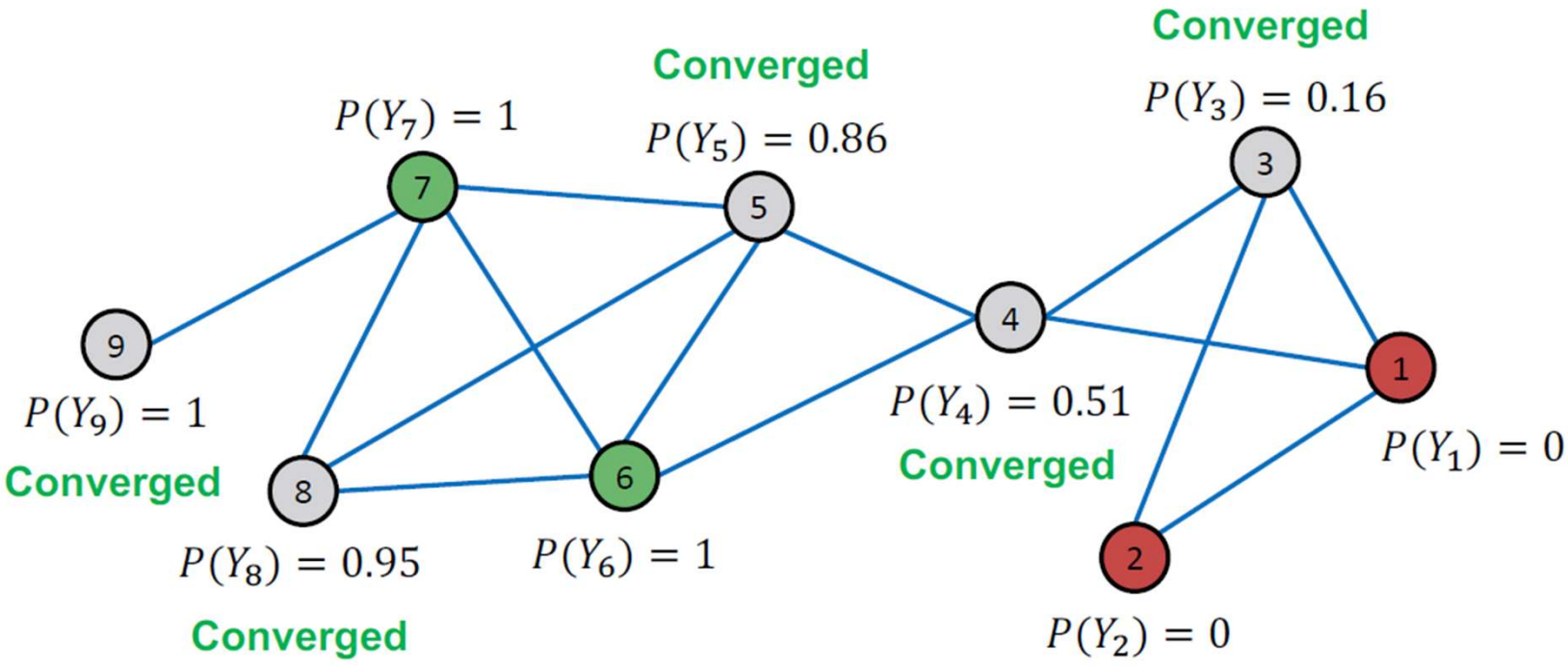


Example: After 4th Iteration



Example: After Convergence

- All scores stabilize after 4 iterations. We therefore predict:
  - **Nodes 4, 5, 8, 9 belong to class 1** ($P_{Y_v} > 0.5$)
  - **Nodes 3 belong to class 0** ($P_{Y_v} < 0.5$)



Overview

- Node Classification
- Relational Classifiers
- Iterative Classification
- Belief Propagation

Iterative Classification

- Relational classifiers do not use node attributes. How can one leverage them?

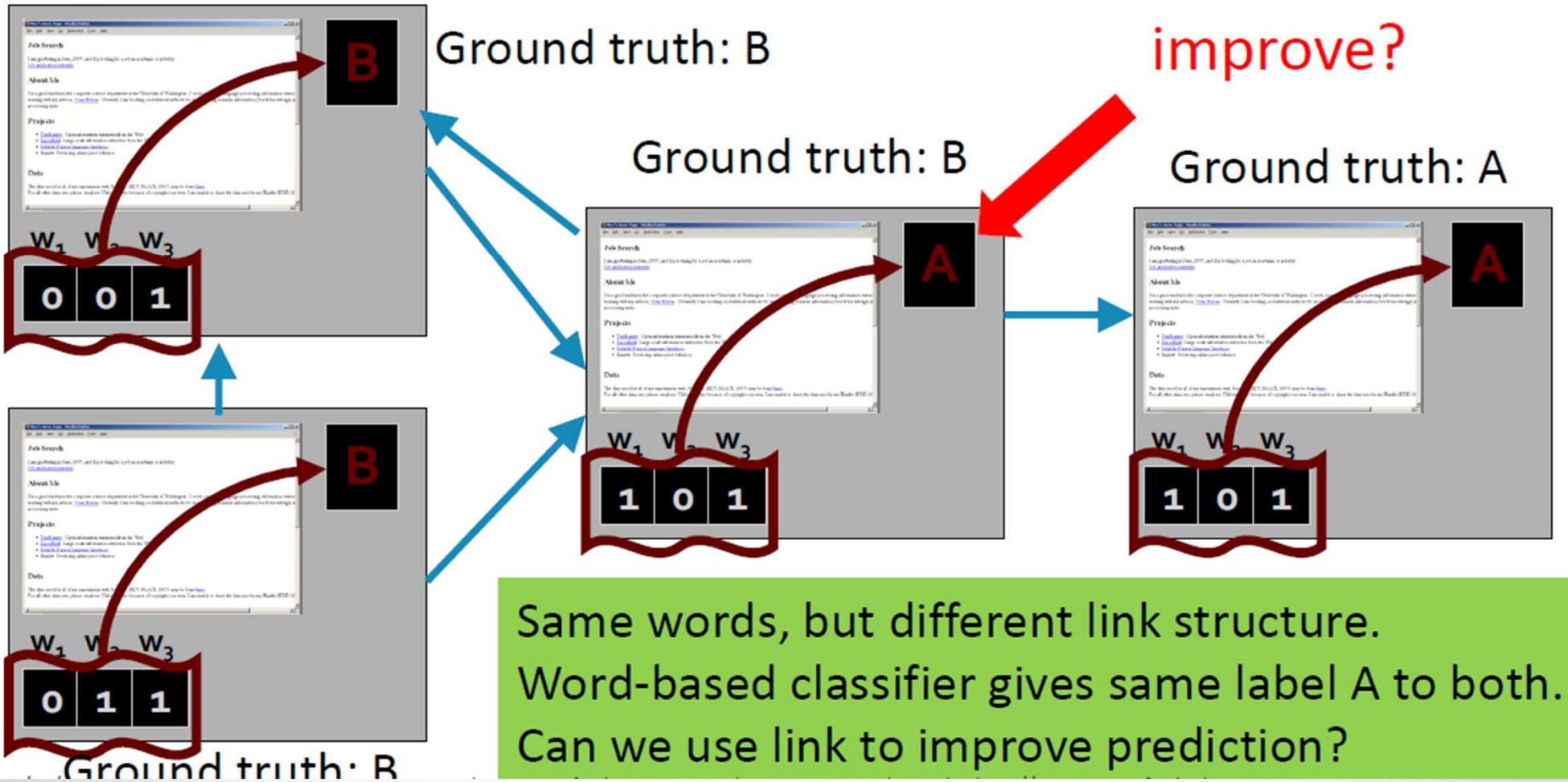
- Main idea of iterative classification: Classify node *i* based on its attributes as well as labels of neighbor set N_i
  - Create a flat vector a_i for each node *i*
  - Train a classifier to classify using a_i
  - Node may have various number of neighbors, so we can aggregate using: count , mode, proportion, mean, exists, etc.

Iterative Classifiers – Basic Architecture

- **Bootstrap phase**
  - Convert each node *i* to a flat vector a_i
  - Use local classifier $f(a_i)$ (e.g., SVM, kNN, …) to compute best value for Y_i
- **Iteration phase:** Iterate till convergence
  - Repeat for each node *i*
    - Update node vector a_i
    - Update label Y_i to $f(a_i)$. This is a hard assignment
  - Iterate until class labels stabilize or max number of iterations is reached
- Note: Convergence is not guaranteed
  - Run for *max* number of iterations

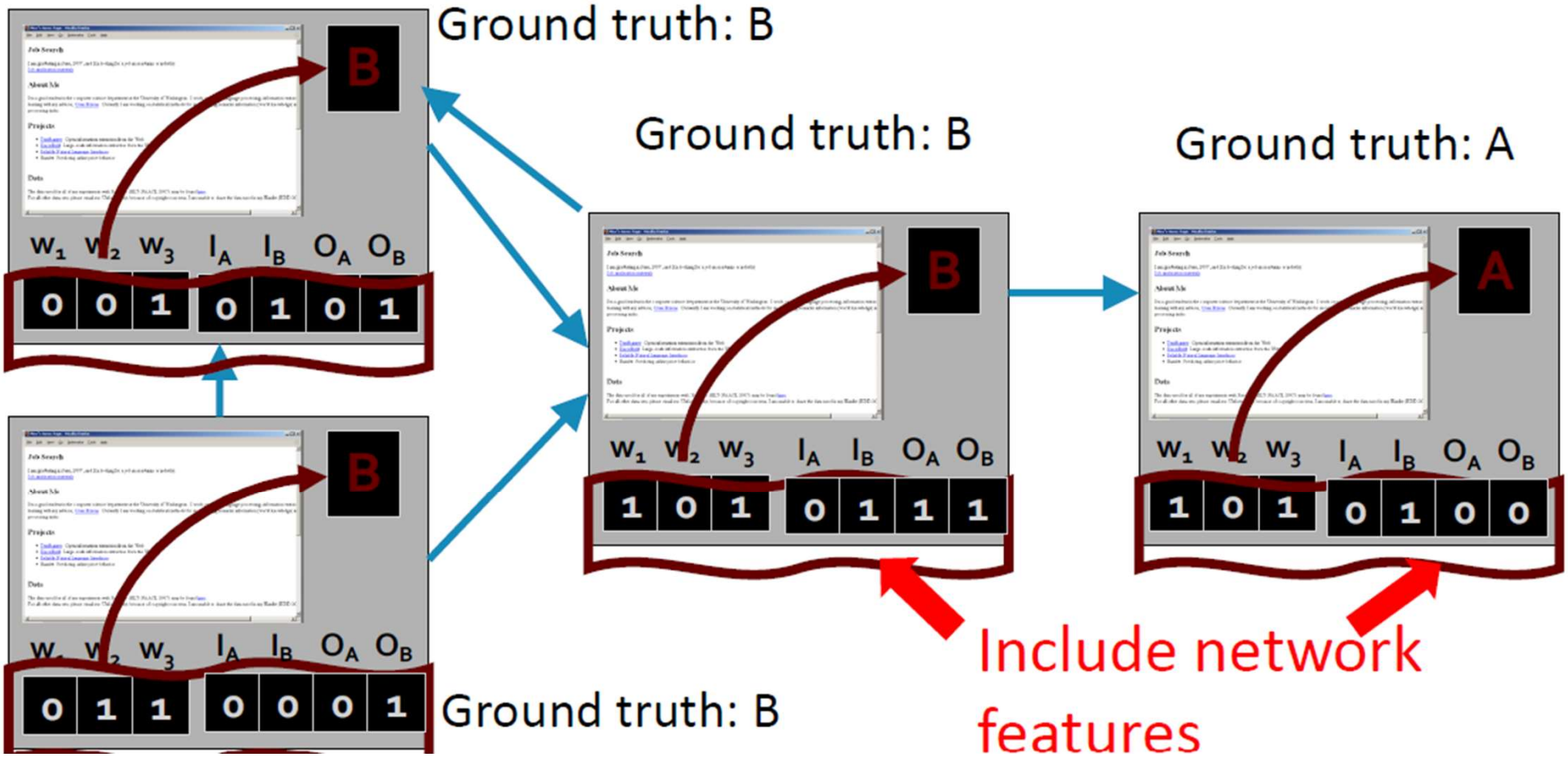
Example: Web Page Classification

- w_1, w_2, w_3, … represent presence of words
- Baseline: train a classifier (e.g., k-NN) to classify pages based on words



Example: Web Page Classification

- Each node maintains a vector of neighborhood labels: (I_A, I_B, O_A, O_B). I = In, O = Out
- I_A = 1 if at least one of the incoming pages is labelled A. Similar definitions for I_B, O_A, and O_B

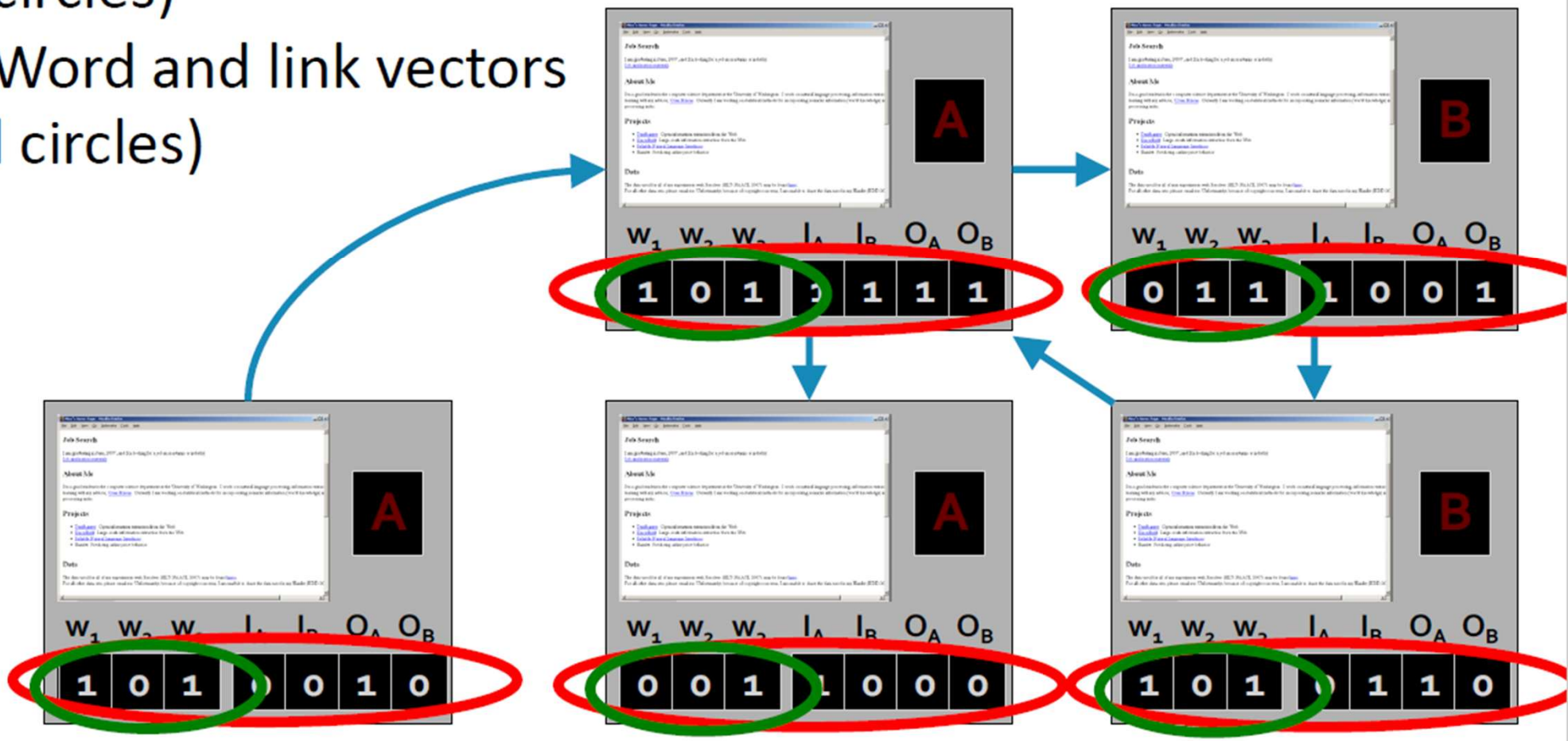


Training set

1. Train
2. Bootstrap
3. Iterate
   a. Update relational features
   b. Classify

On a different **training set**, train two classifiers:

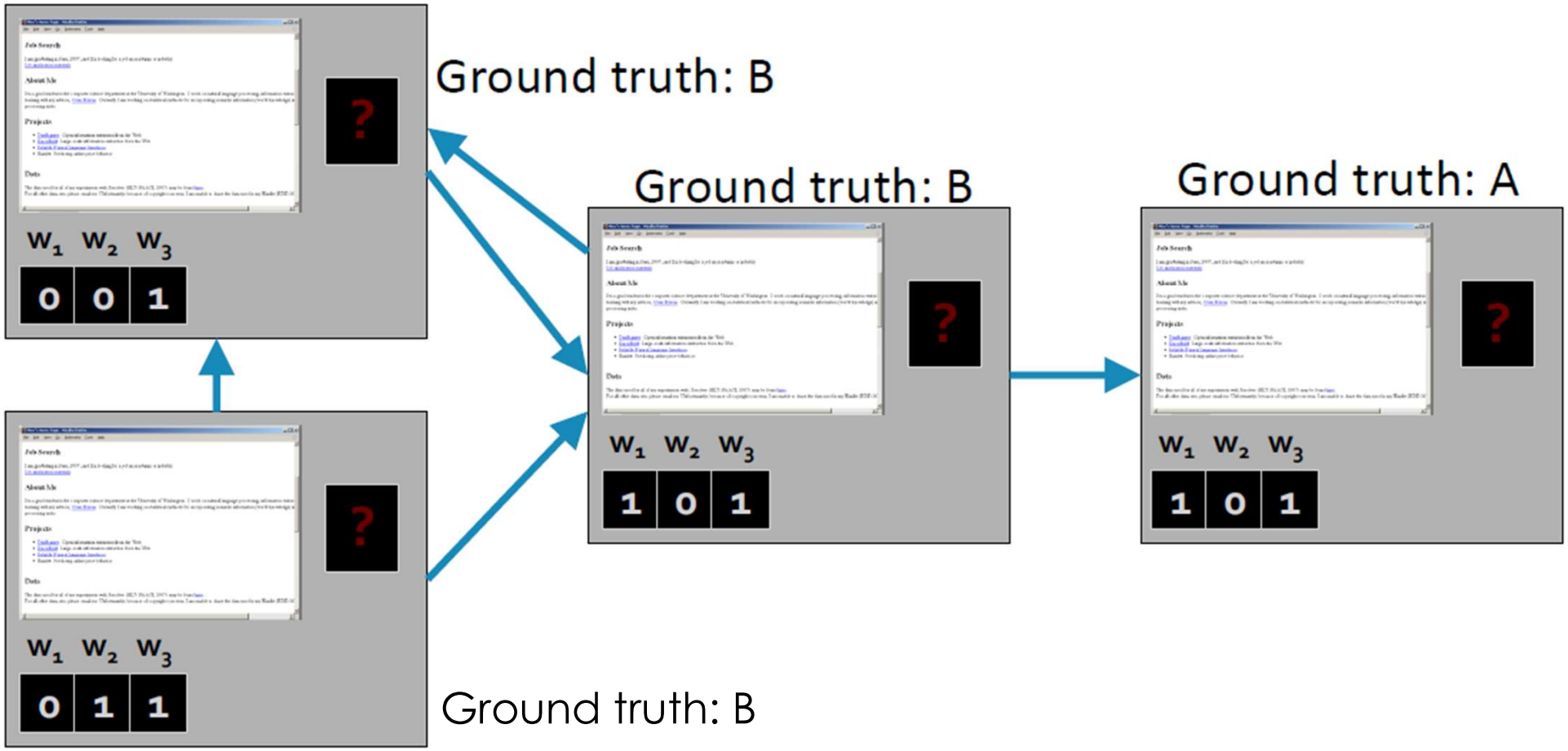
1. Word vector only (green circles)
2. Word and link vectors (red circles)



On test set

1. Train
2. Bootstrap
3. Iterate
   a. Update relational features
   b. Classify

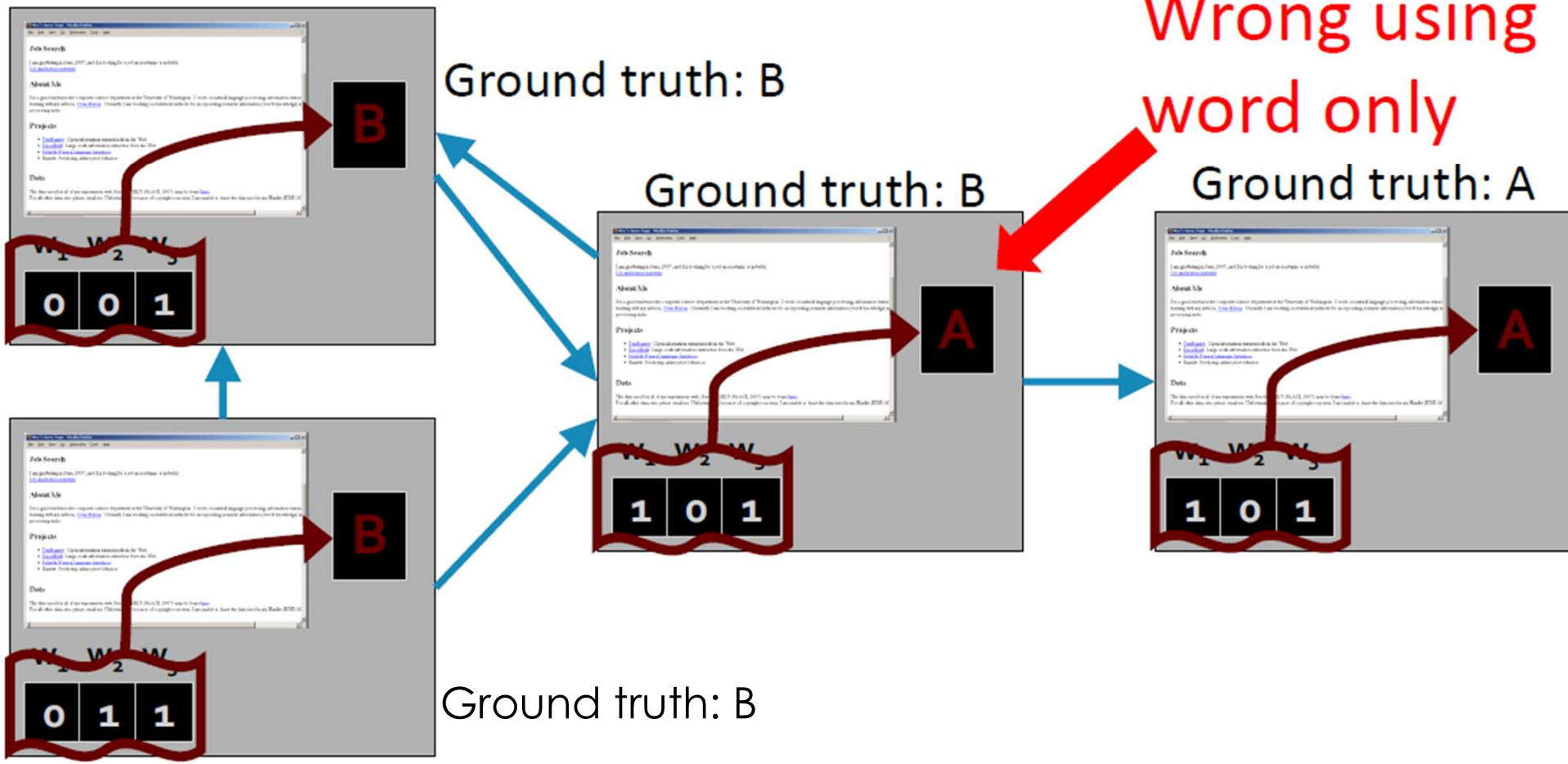
Use trained word-vector classifier to bootstrap on test set



On test set

1. Train
2. Bootstrap
3. Iterate
   a. Update relational featur
   b. Classify

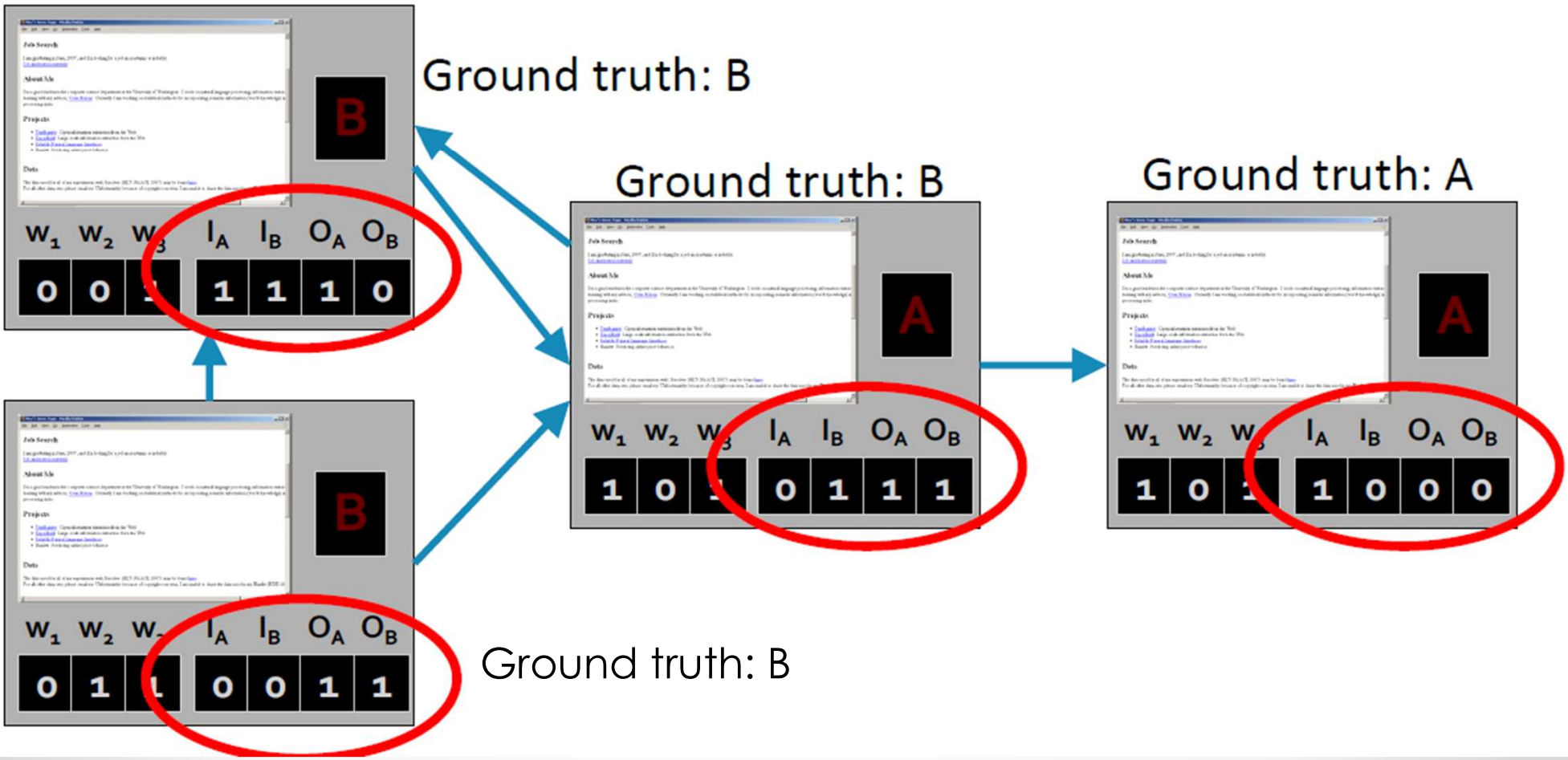
Use trained word-vector classifier to bootstrap on test set



On test set

1. Train
2. Bootstrap
3. Iterate
   - a. Update relational features
   - b. Classify

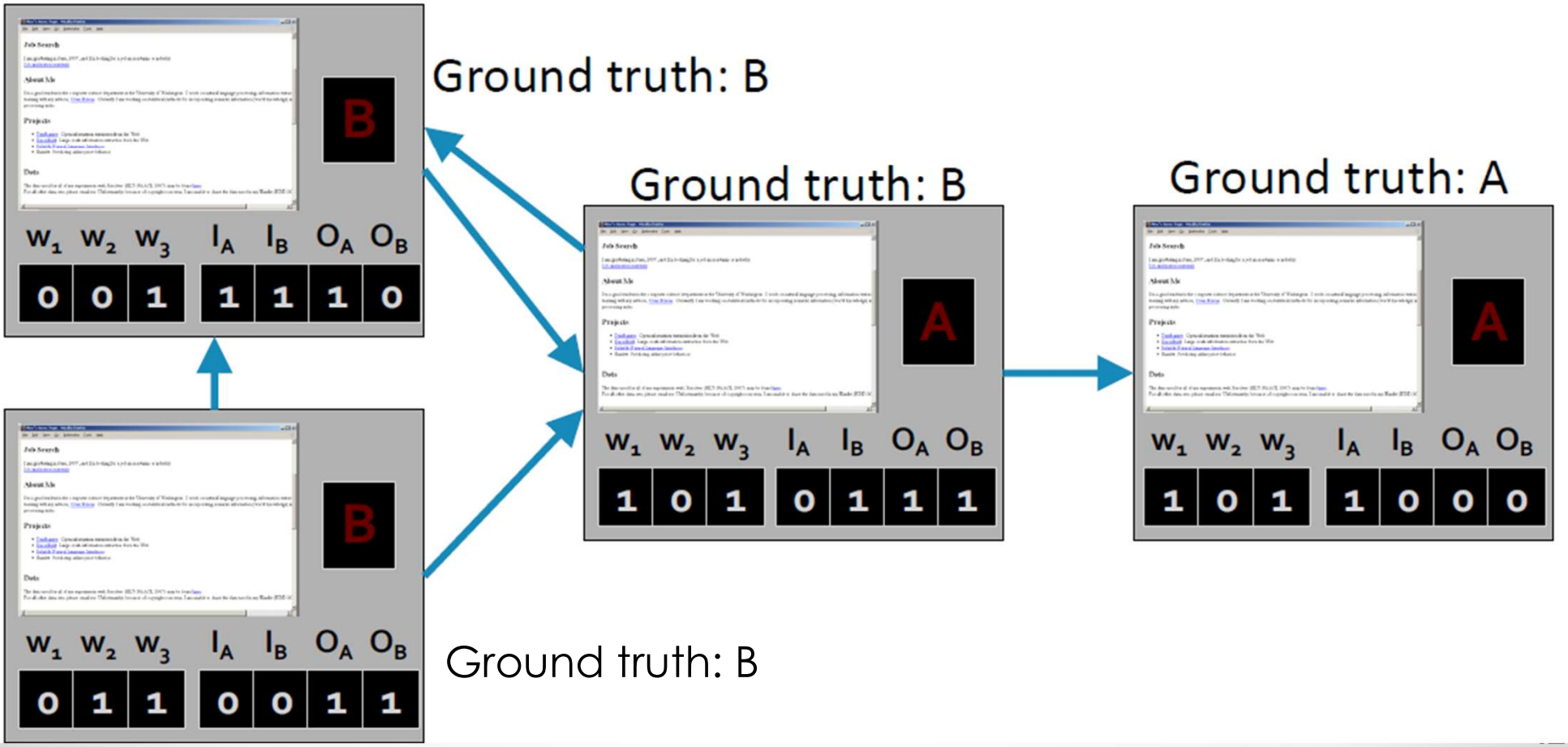
Update neighborhood vector for all nodes



On test set

1. Train
2. Bootstrap
3. Iterate
   a. Update relational features
   b. Classify

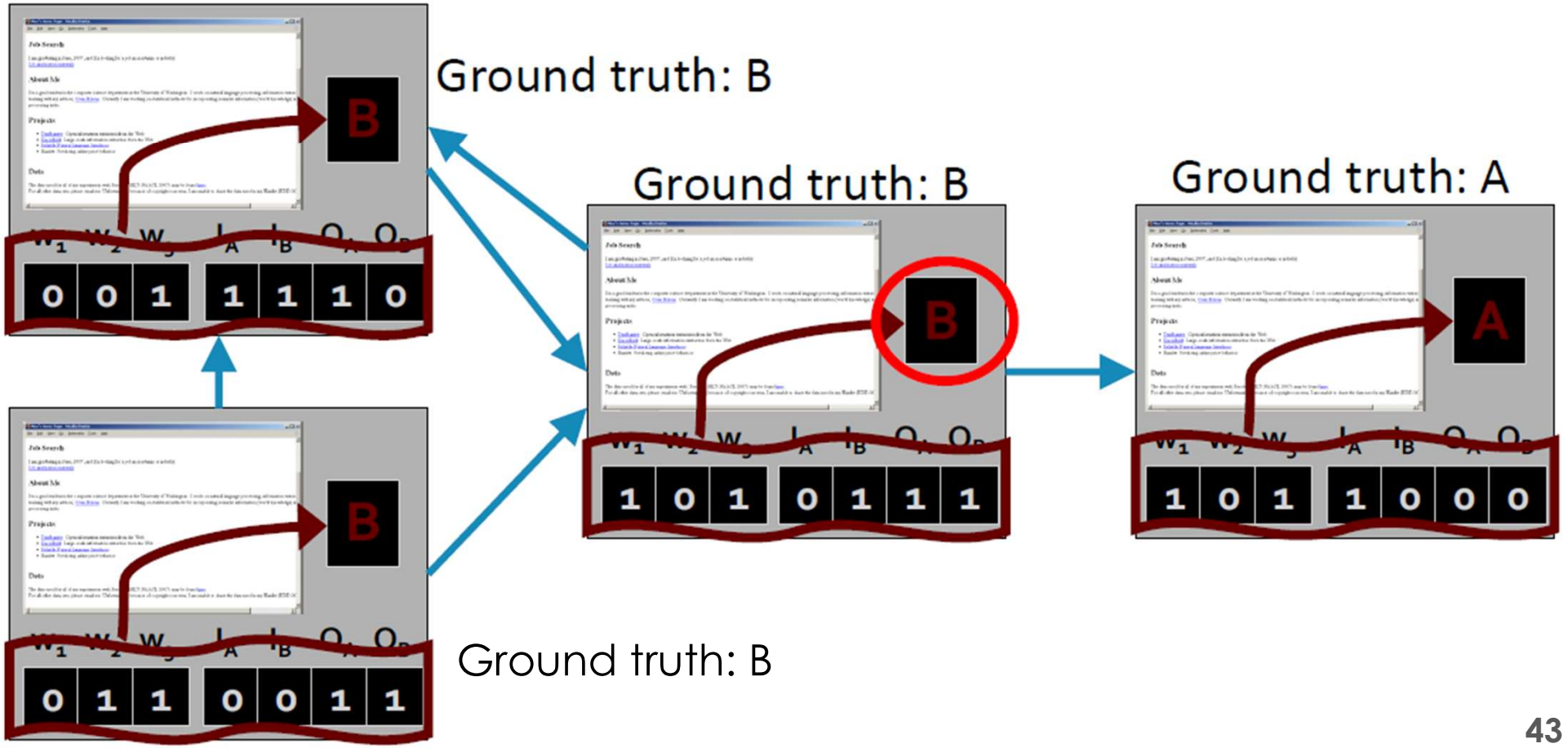
Reclassify all nodes



On test set

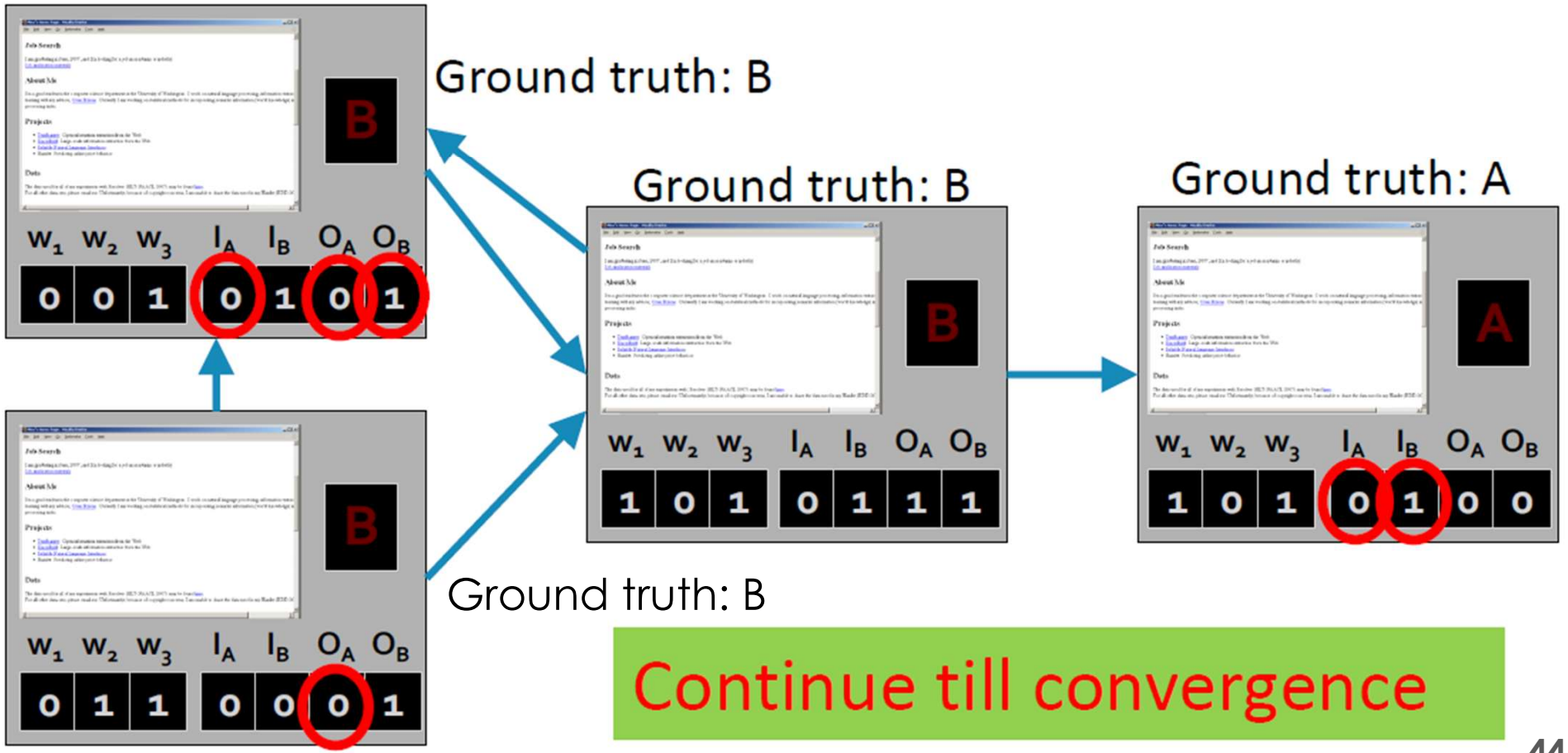
1. Train
2. Bootstrap
3. Iterate
   a. Update relational features
   b. Classify

Reclassify all nodes



On test set

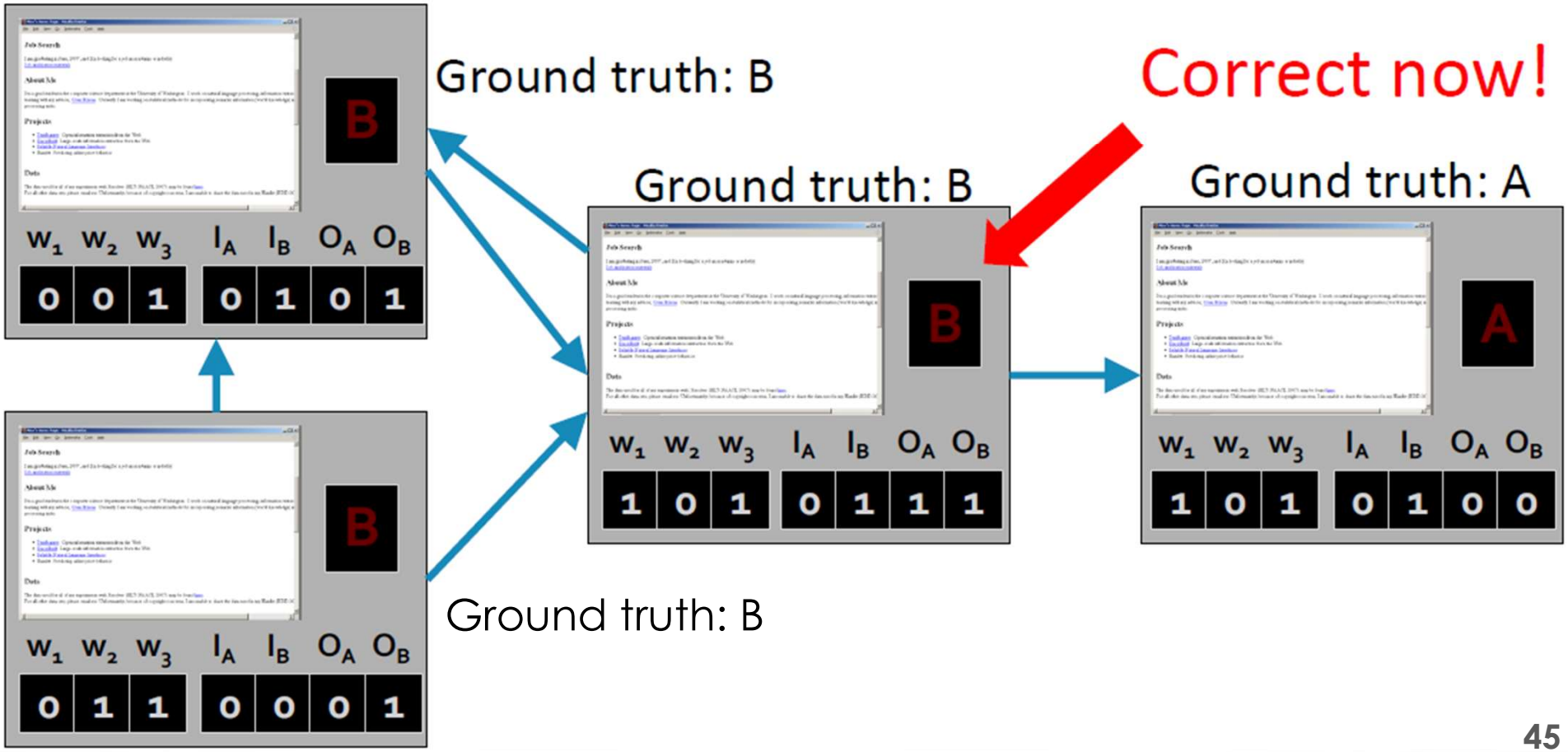
1. Train
2. Bootstrap
3. Iterate
   - a. Update relational features
   - b. Classify



Continue till convergence

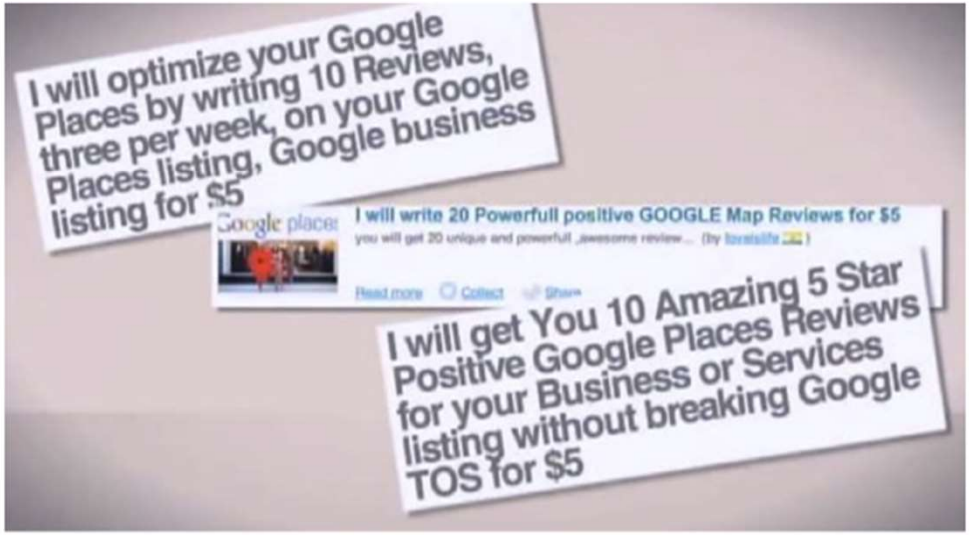
Final

1. Train
2. Bootstrap
3. Iterate
   a. Update relational features
   b. Classify



Example: Fake Reviewer/Review Detection

- Review sites are an attractive target for spam: a +1 star increase in rating increases revenue by 5-9%!
- Often hype/defame spam
- Paid spammers



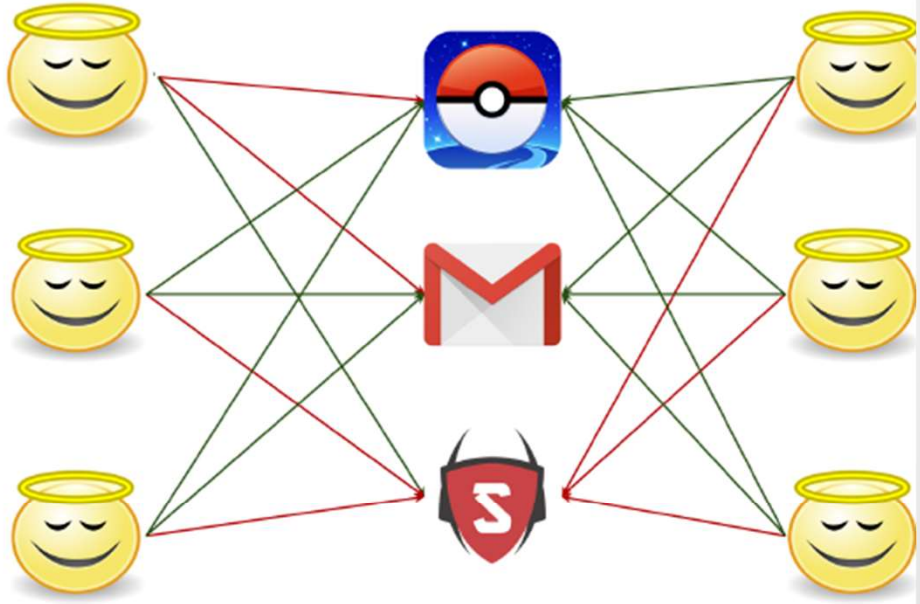
Reference: S. Kumar, et al. REV2: Fraudulent User Prediction in Rating Platforms, Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, 2018.

Fake Review Spam Detection

- Behavioral analysis
  - Individual features, geographic locations, login times, session history, etc.
- Language analysis
  - Use of superlatives, lots of self-referencing, rate of misspellings, many agreement words, …
- Easy to fake: **individual behaviors, content of review**
- Hard to fake: **graph structure**
  - Graphs capture relationships between reviewers, reviews, stores

Problem Setup

- **Input**: bipartite rating graph as a weighted signed network:
  - Nodes: users, products
  - Edges: rating scores between -1 and +1

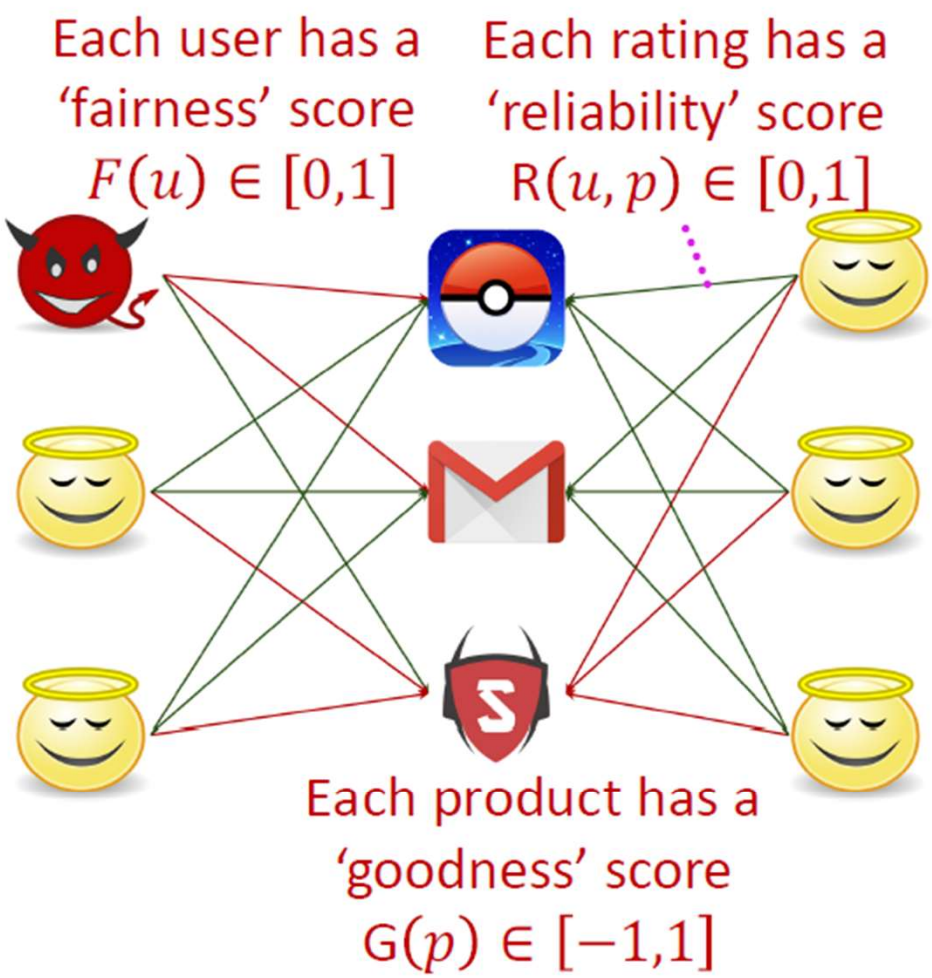


- **Output**: set of users that give fake ratings

Red edges = -1 rating
Green edges = +1 rating

Solution Formulation

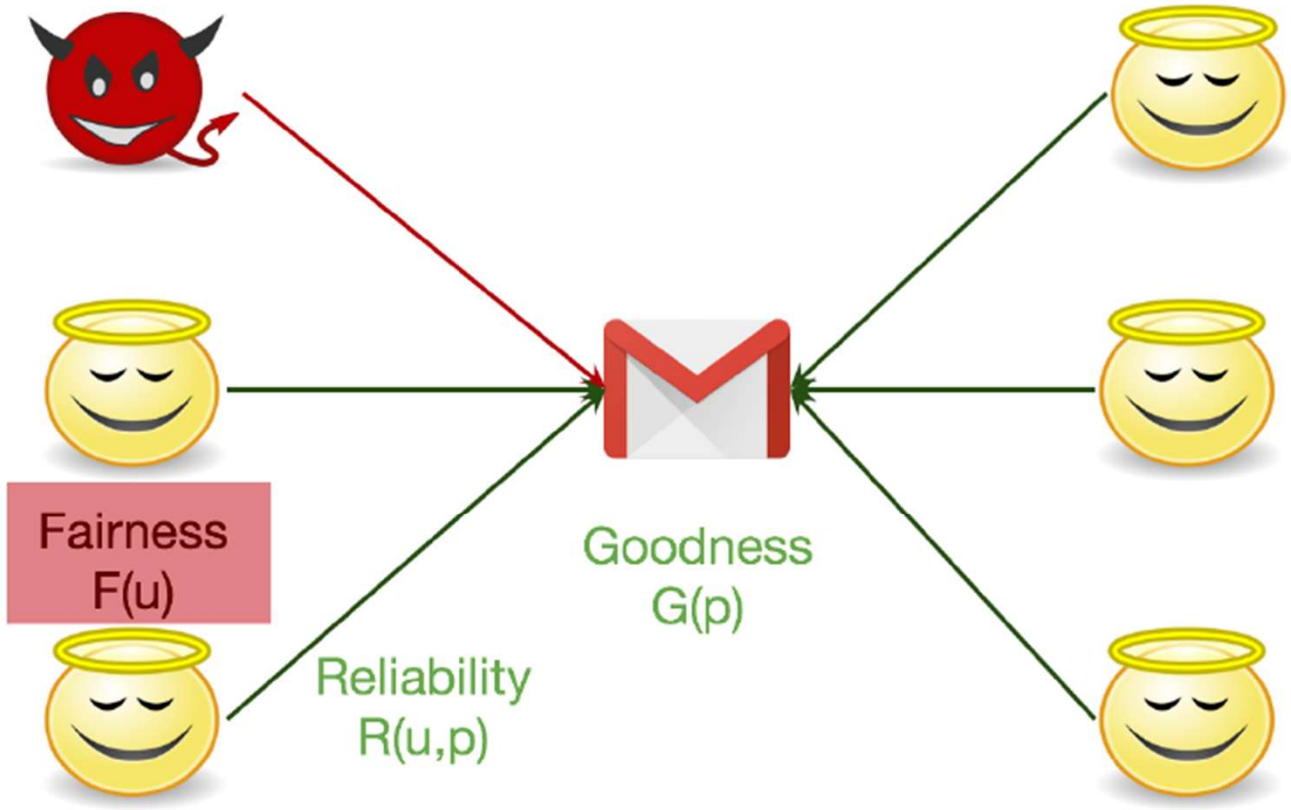
- **Basic idea**: Users, products, and ratings have intrinsic quality scores:
  - Users have fairness scores
  - Products have goodness scores
  - Ratings have reliability scores
- All values are unknown
- How can one calculate the values for all nodes and edges simultaneously?
- **Solution: Iterative classification**



Fairness of users

- Fixing goodness and reliability, fairness is updated as:

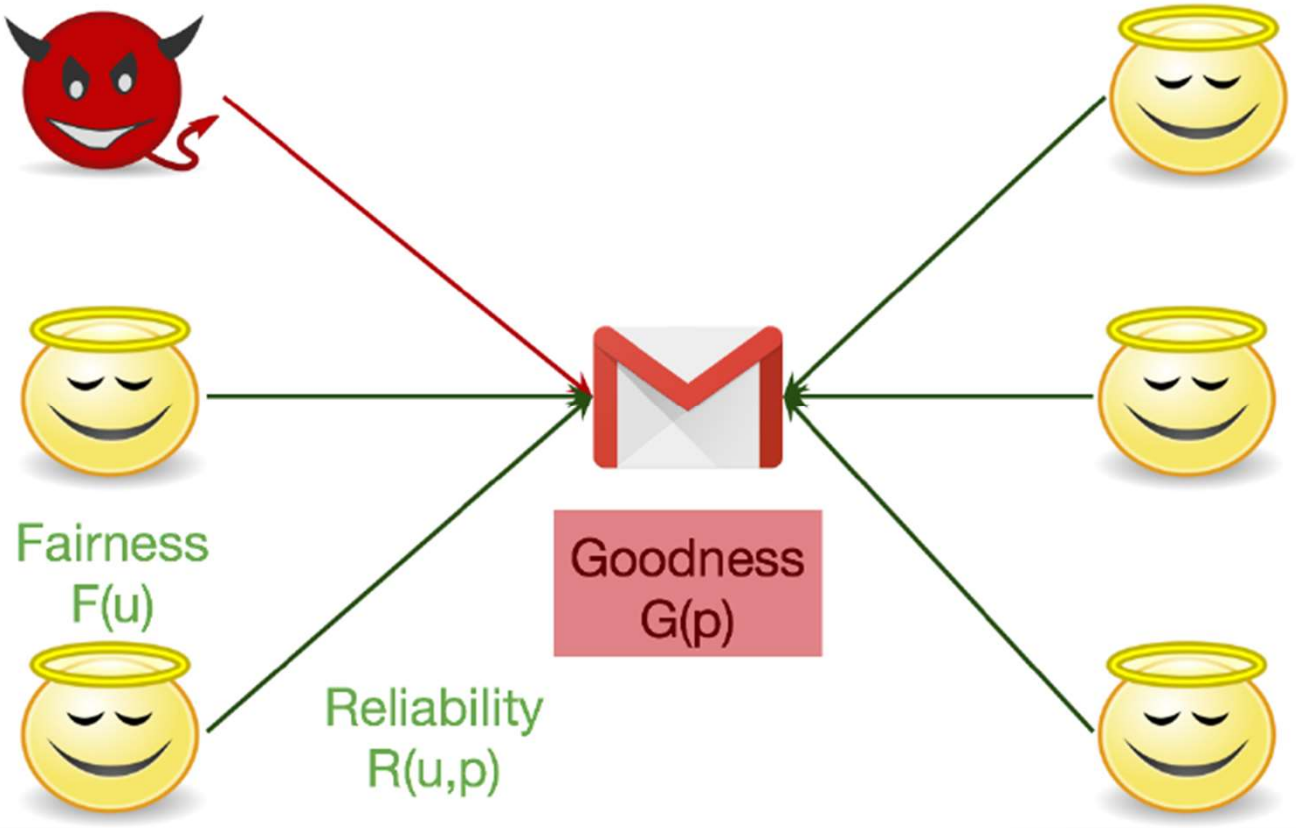
$$F(u) = \frac{\sum_{(u,p)\in \mathrm{Out}(u)} R(u,p)}{|\mathrm{Out}(u)|}$$



Goodness of products

- Fixing fairness and reliability, goodness is updated as:

$$G(p) = \frac{\sum_{(u,p)\in In(p)} R(u,p) \cdot score(u,p)}{|In(p)|}$$



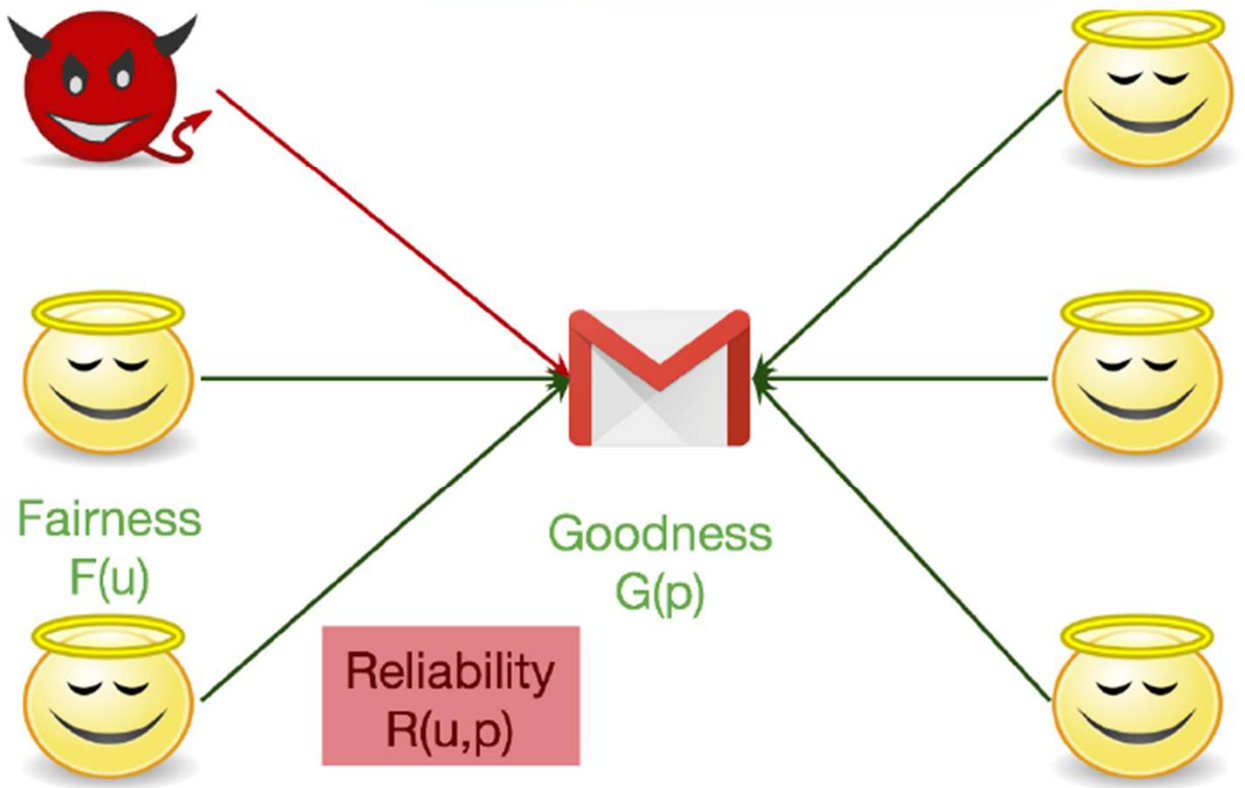
Reliability of ratings

- Fixing fairness and goodness, reliability is updated as:

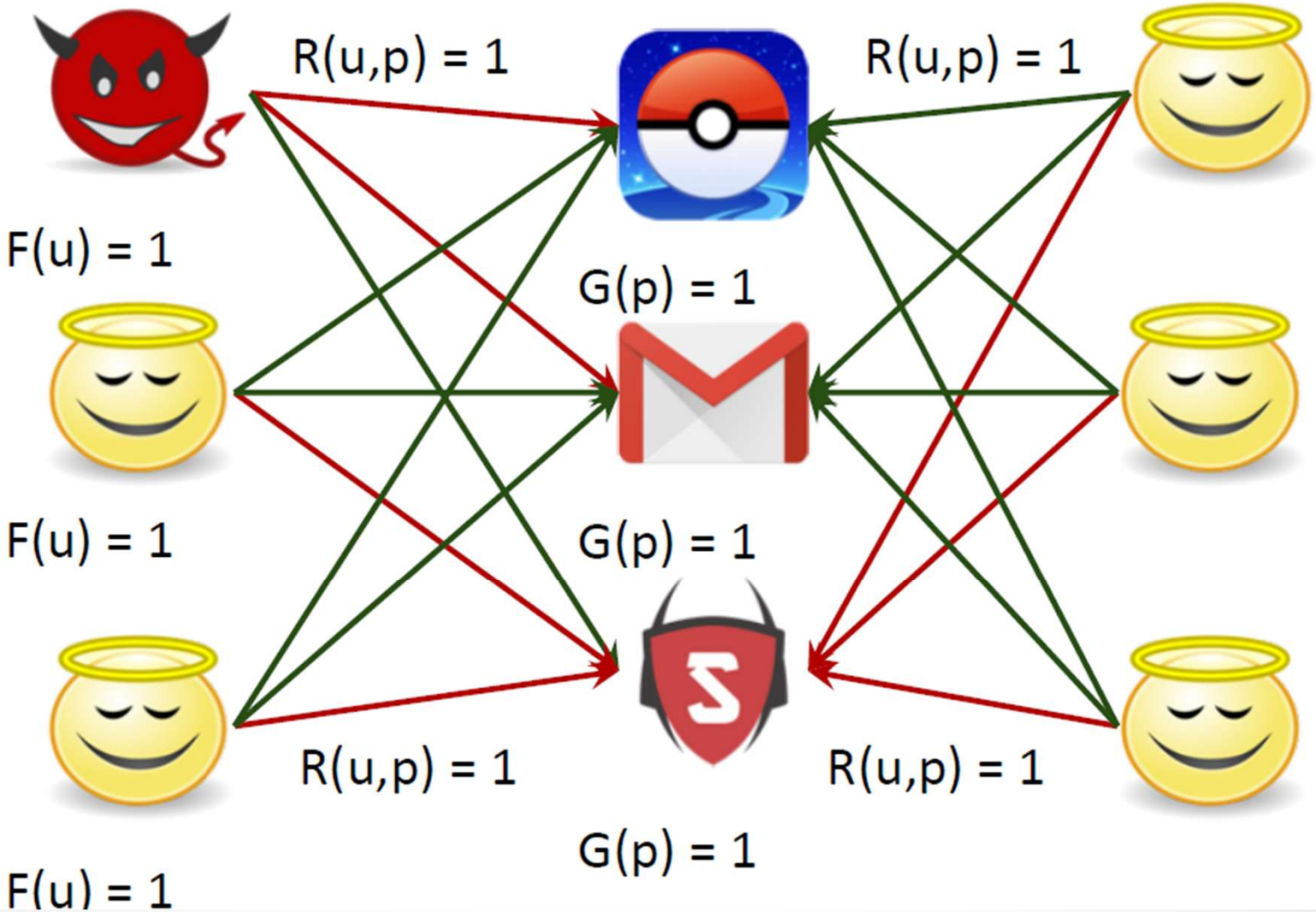
How fair is the user who gives the rating

How close is the rating from the goodness of product

$$R(u,p) = \frac{1}{\gamma_1 + \gamma_2}(\gamma_1 \cdot F(u) + \gamma_2 \cdot (1 - \frac{|\text{score}(u,p) - G(p)|}{2}))$$

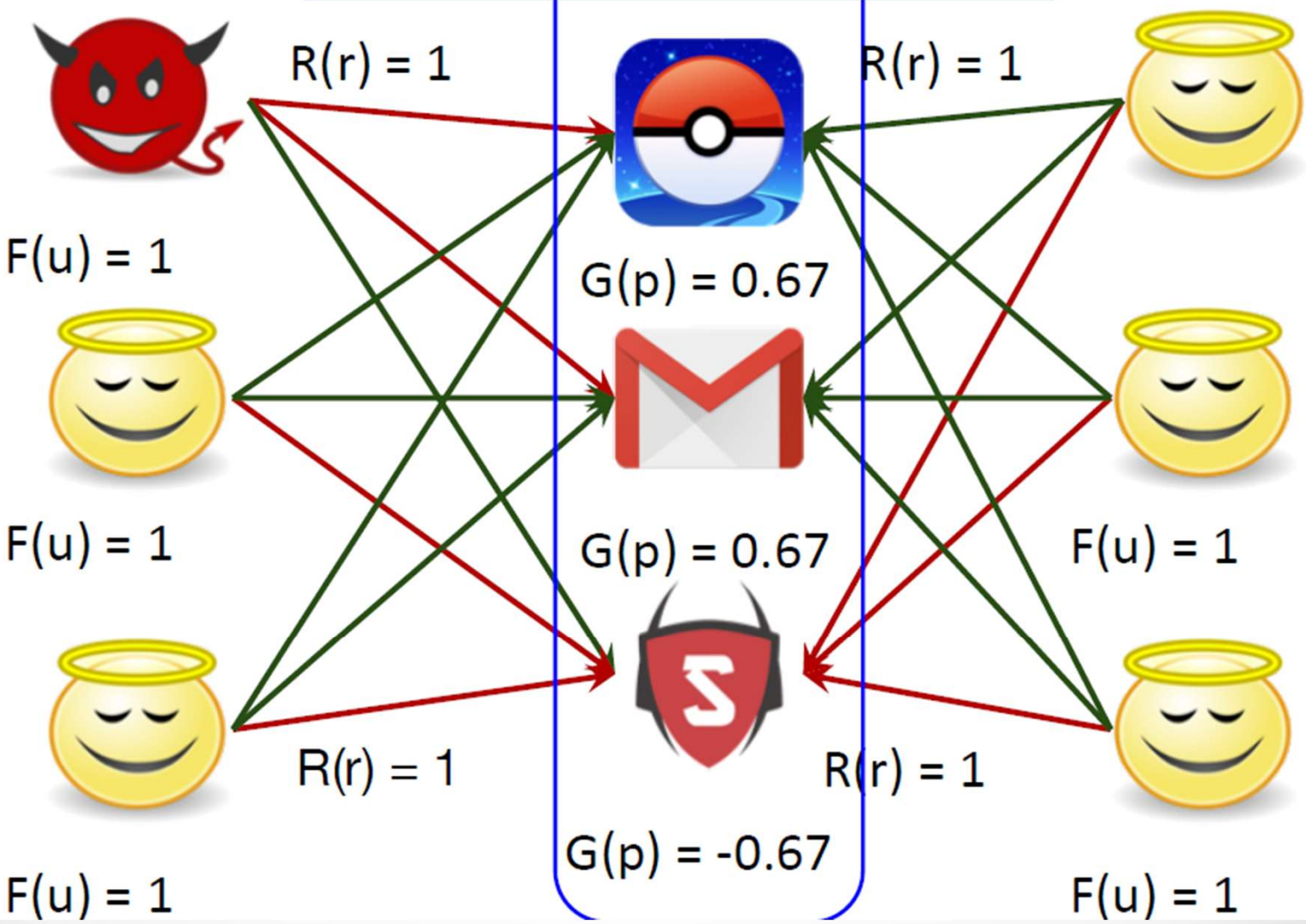


Initialization to best scores



Updating Goodness – Iteration 1

$$G(p) = \frac{\sum_{(u,p)\in In(p)} R(u,p) \cdot score(u,p)}{|In(p)|}$$



Updating Reliability – Iteration 1

$$R(u,p) = \frac{1}{\gamma_1 + \gamma_2}(\gamma_1 \cdot F(u) + \gamma_2 \cdot (1 - \frac{|\text{score}(u,p) - G(p)|}{2}))$$

Both gamma values are set to 1









R(r) = 0.58

R(r) = 0.58

F(u) = 1

G(p) = 0.67

R(r) = 0.92

F(u) = 1

F(u) = 1

G(p) = 0.67

F(u) = 1

R(r) = 0.92

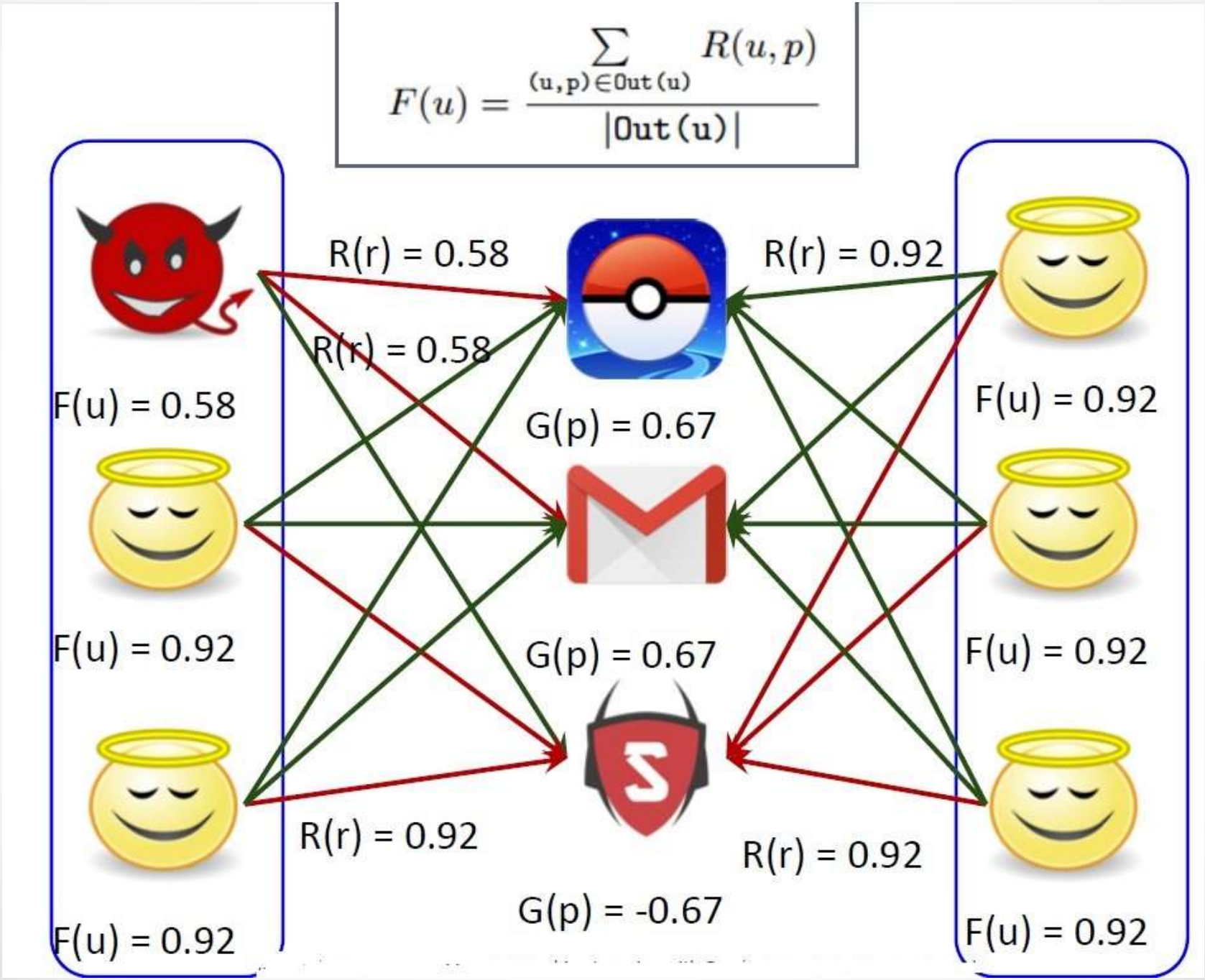
G(p) = -0.67

R(r) = 0.92

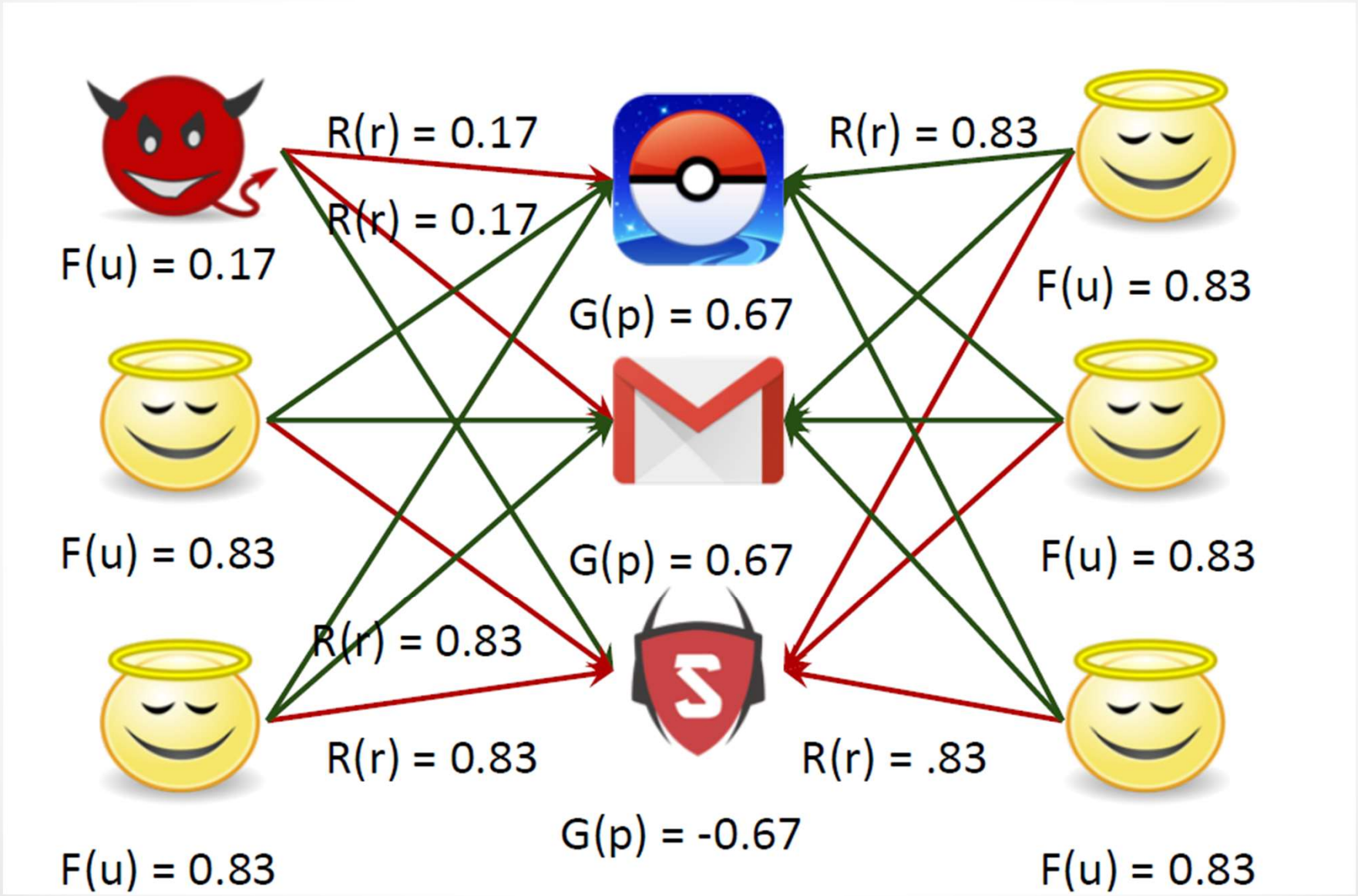
F(u) = 1

Updating Fairness – Iteration 1

$$F(u) = \frac{\sum_{(u,p) \in Out(u)} R(u,p)}{|Out(u)|}$$

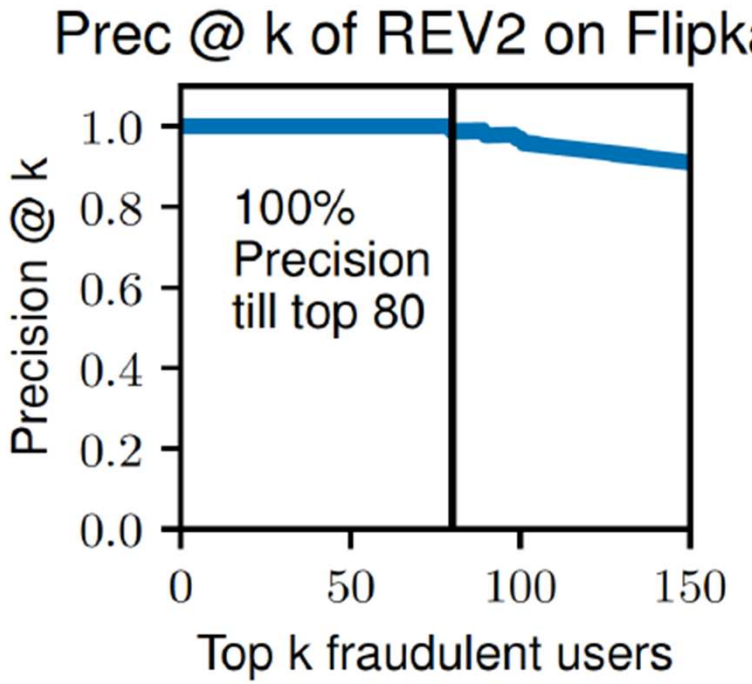


After Convergence



Performance

- Low fairness users = Fraudsters
- 127 of 150 lowest fairness users in Flipkart were real fraudsters



Properties of solution

- Guaranteed to converge
- Number of iterations till convergence is upper-bounded
- Time–complexity: linear in the number of edges in the graph

Overview

- Node Classification
- Relational Classifiers
- Iterative Classification
- Belief Propagation

Loopy Belief Propagation

- Belief Propagation is a dynamic programming approach to **answering probability queries in a graph** (e.g. probability of node v belonging to class 1)
- Iterative process in which neighbor nodes "talk" to each other, **passing messages**

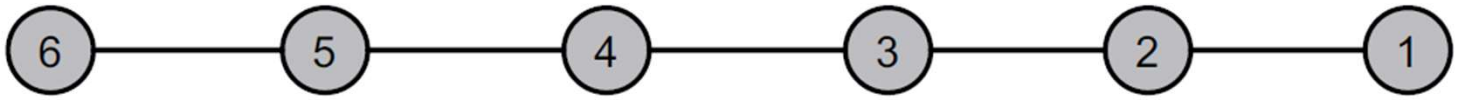
"I (node v) believe you (node u) belong to class 1 with likelihood …"

- **When consensus is reached, calculate final belief**

Message Passing: Basics

- **Task**: Count the number of nodes in a graph*
- **Condition:** Each node can only interact (pass message) with its neighbors

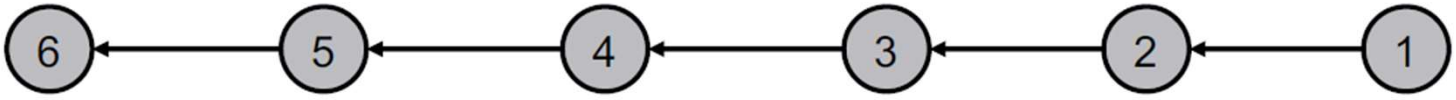
- **Example**: path graph



* Potential issues when the graph contains cycles.
We'll get back to it later!

Message Passing: Algorithm

- **Task**: Count the number of nodes in a graph
- **Algorithm**:
  - Define an ordering of nodes (that results in a path)
  - Edge directions are according to order of nodes
    - Edge direction defines the order of message passing
  - For node i from 1 to 6
    - Compute the message from node i to $i + 1$ (number of nodes counted so far)
    - Pass the message from node i to $i + 1$



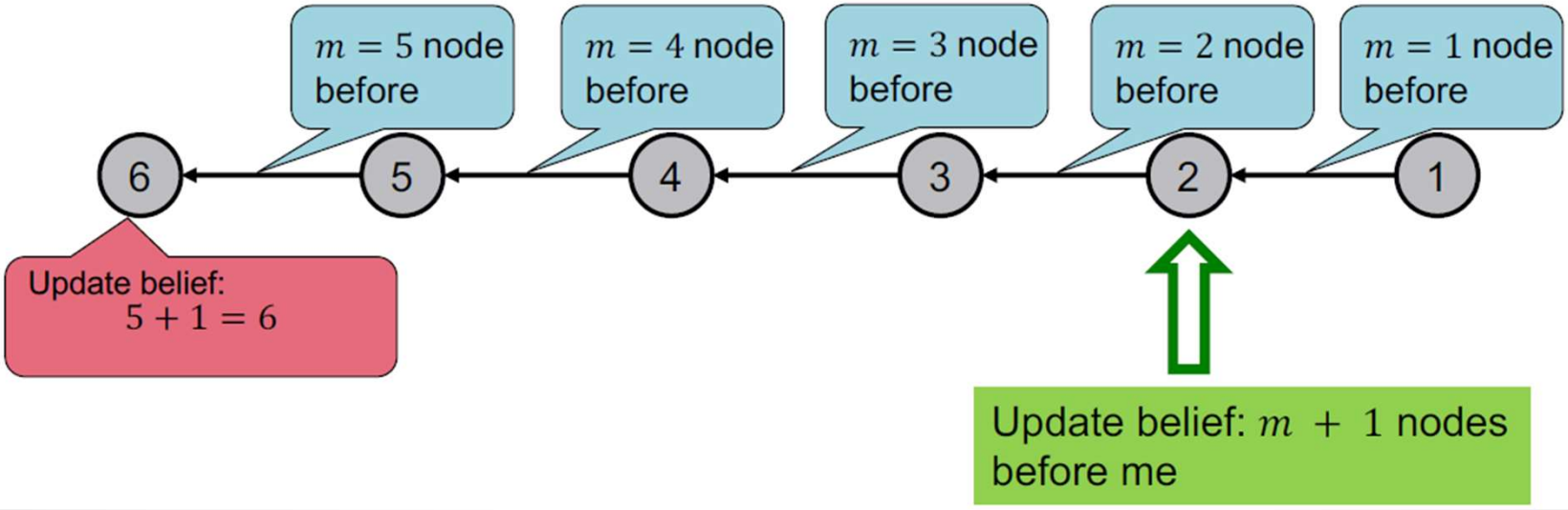
Message Passing: Basics

Task: Count the number of nodes in a graph

Condition: Each node can only interact (pass message) with its neighbors

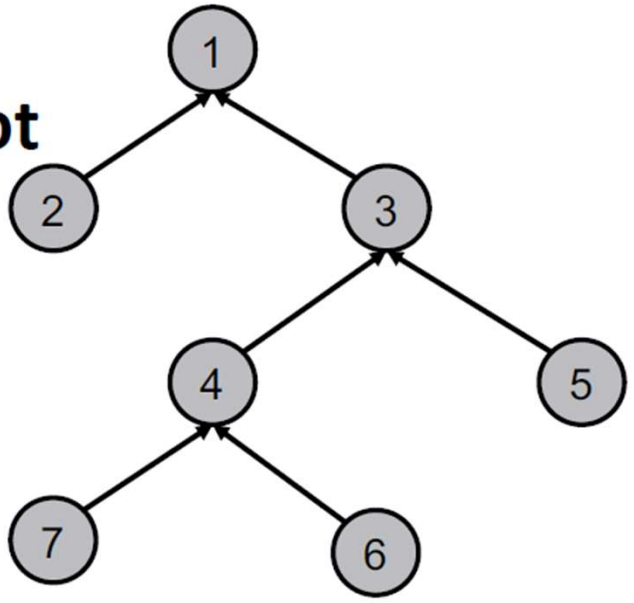
Solution: Each node listens to the message from its neighbor, updates it, and passes it forward

m: the message

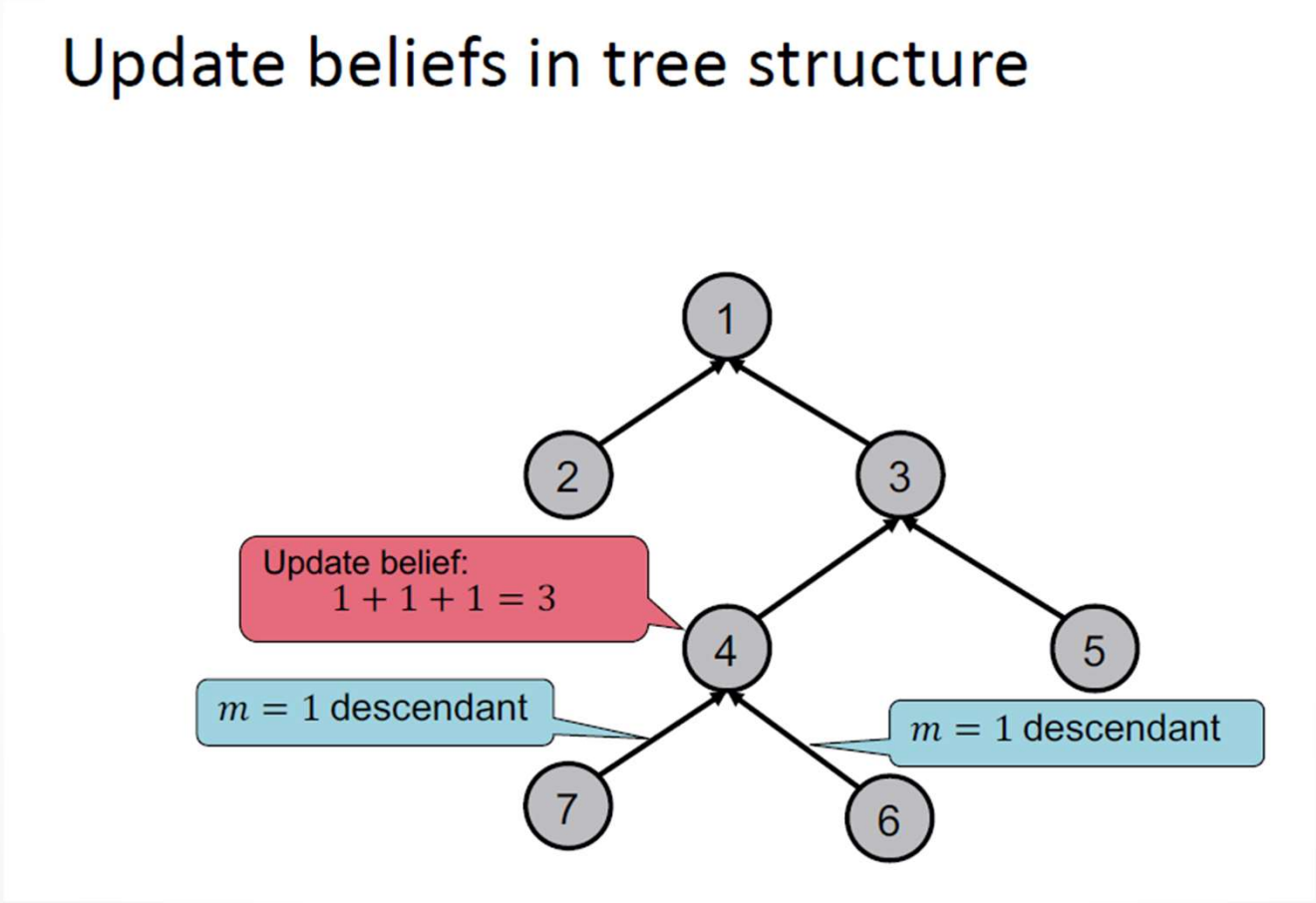


Generalizing to a Tree

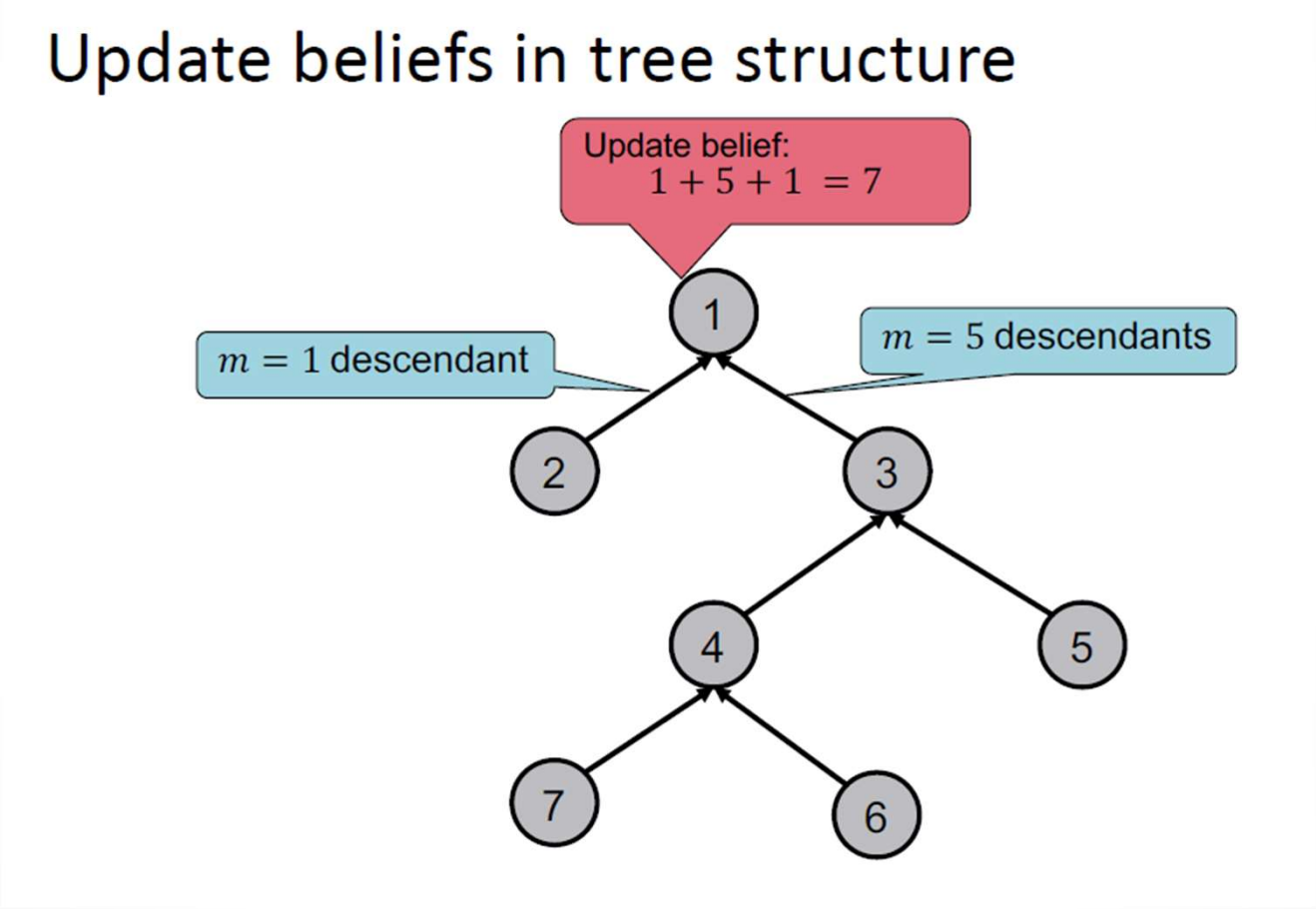
- We can perform message passing not only on a path graph, but also on a tree-structured graph
- Define order of message passing from **leaves** to **root**



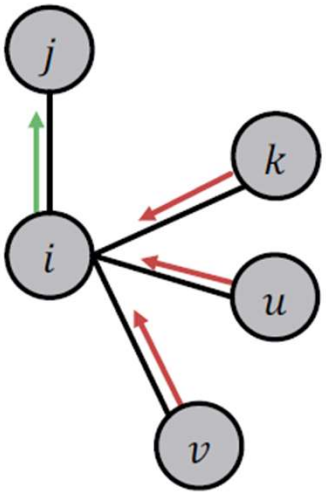
Message Passing in a Tree



Message Passing in a Tree



Loopy BP Algorithm



What message will i send to j?

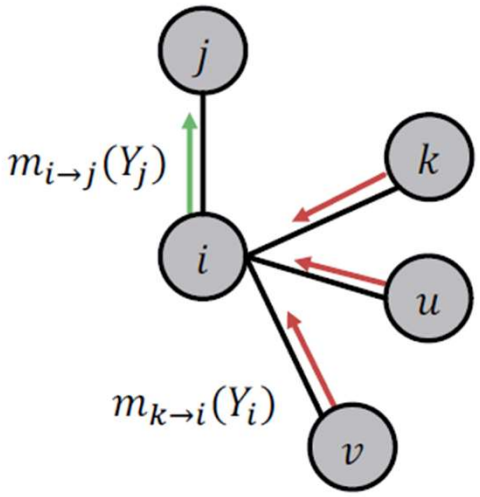
- It depends on what i hears from its neighbors
- Each neighbor passes a message to i its beliefs of the state of i

I (node i) believe that you (node j) belong to class Y_j with probability …

Notation

- ***Label-label potential matrix*** ψ : Dependency between a node and its neighbor. $\boldsymbol{\psi}(Y_i, Y_j)$ is proportional to the probability of a node j being in class Y_j given that it has neighbor i in class Y_i.
- ***Prior belief*** ϕ : $\phi(Y_i)$ is proportional to the probability of node i being in class Y_i.
- $m_{i \to j}(Y_j)$ is i's message / estimate of j being in class Y_j.
- $\mathcal{L}$ is the set of all classes/labels

Loopy BP Algorithm



1. Initialize all messages to 1
2. Repeat for each node:

All messages sent by neighbors from previous round

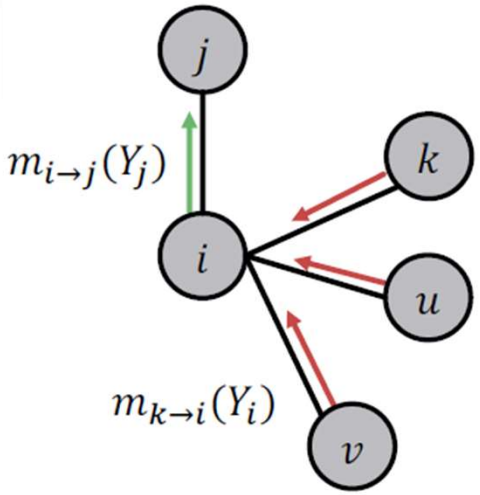
Label-label potential

$$m_{i\to j}(Y_j) = \sum_{Y_i\in\mathcal{L}} \psi(Y_i, Y_j)\,\phi_i(Y_i) \prod_{k\in N_i\setminus \mathrm{j}} m_{k\to i}(Y_i), \forall Y_j \in \mathcal{L}$$

Sum over all states

Prior

Loopy BP Algorithm



After convergence:

$b_i(Y_i)$ = node i's belief of being in class Y_i

All messages from neighbors

Prior

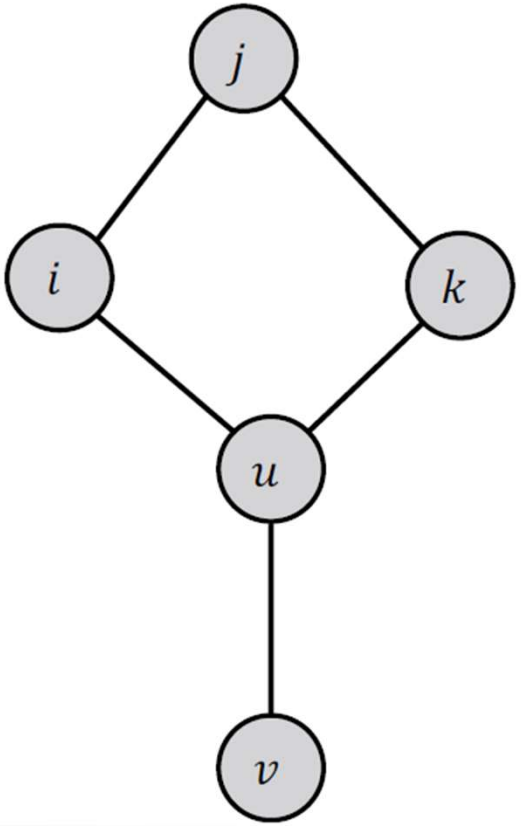
$$b_i(Y_i) = \phi_i(Y_i) \prod_{j \in N_i} m_{j \to i}(Y_i), \quad \forall Y_i \in \mathcal{L}$$

Example: Loopy Belief Propagation

- Now we consider a graph with cycles
- There is no longer an ordering of nodes
- We apply the same algorithm as in previous slides:
  - Start from arbitrary nodes
  - Follow the edges to update the neighboring nodes

Example: Loopy Belief Propagation

What if our graph has cycles?

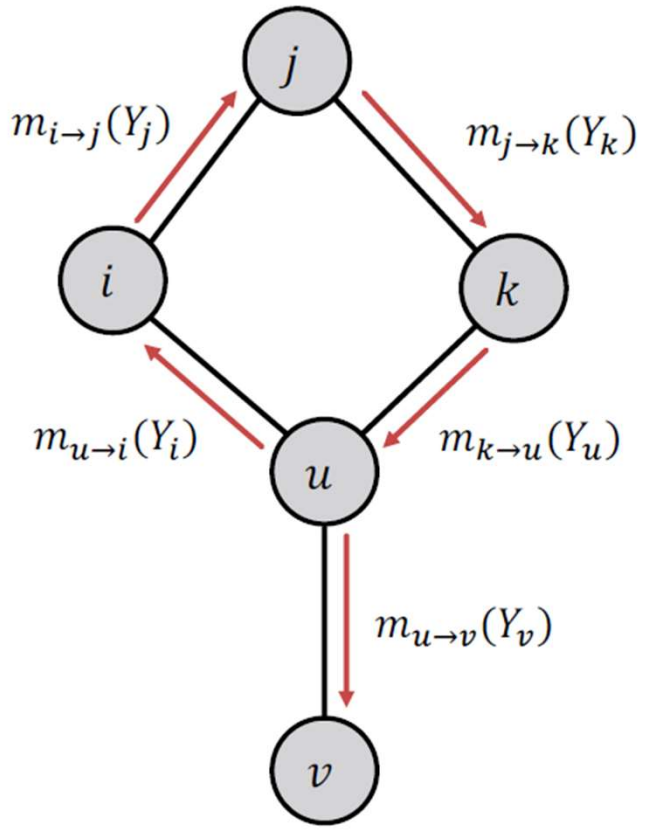


Messages from different subgraphs are **no longer independent**!

But we can still run BP, but it will pass messages in loops.

Example: Loopy Belief Propagation

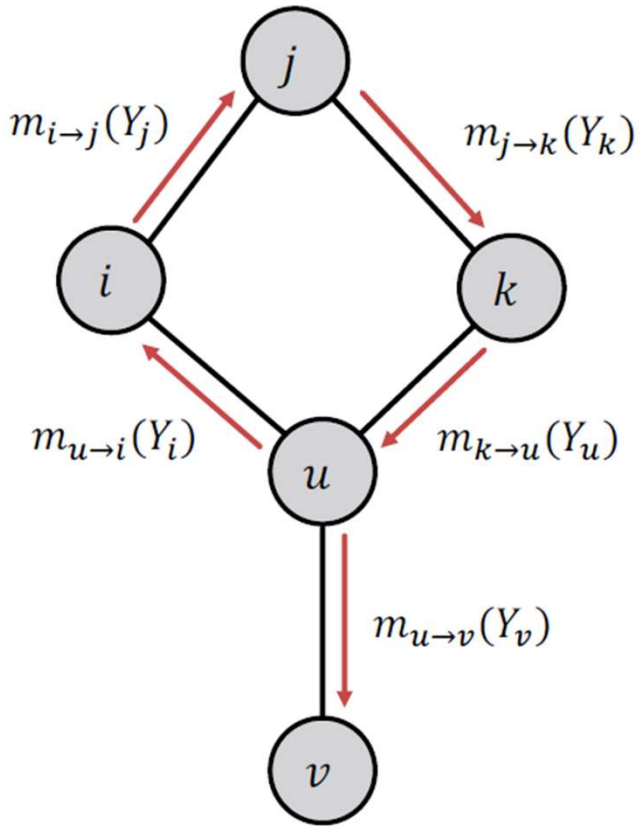
What if our graph has cycles?



Messages from different subgraphs are **no longer independent**!

But we can still run BP, but it will pass messages in loops.

What can go wrong?



- **Beliefs may not converge**
  - Message $m_{u\to i}(Y_i)$ is based on initial belief of i, not a **separate evidence** for i
  - The initial belief of i (which could be incorrect) is reinforced by the cycle $i \to j \to k \to u \to i$
- However, in practice, Loopy BP is still a good heuristic for complex graphs which contain many branches.

In most real-world graphs, cycles are few or have weak connections

Belief Propagation - Analysis

- **Advantages**:
  - Easy to program & parallelize
  - General: can apply to any graph model with any form of potentials
    - Potential can be higher order: e.g. $\boldsymbol{\psi}(Y_i, Y_j, Y_k, Y_v \ldots)$
- **Challenges**:
  - Convergence is not guaranteed (when to stop), especially if many closed loops
- **Potential functions** (parameters)
  - Require training to estimate

Summary

- Three collective classification algorithms:
  - **Relational models**
    - Weighted average of neighborhood properties
    - Cannot take node attributes while labeling
  - **Iterative classification**
    - Update each node's label using own and neighbor's labels
    - Can consider node attributes while labeling
  - **Belief propagation**
    - Message passing to update each node's belief of itself based on neighbors' beliefs