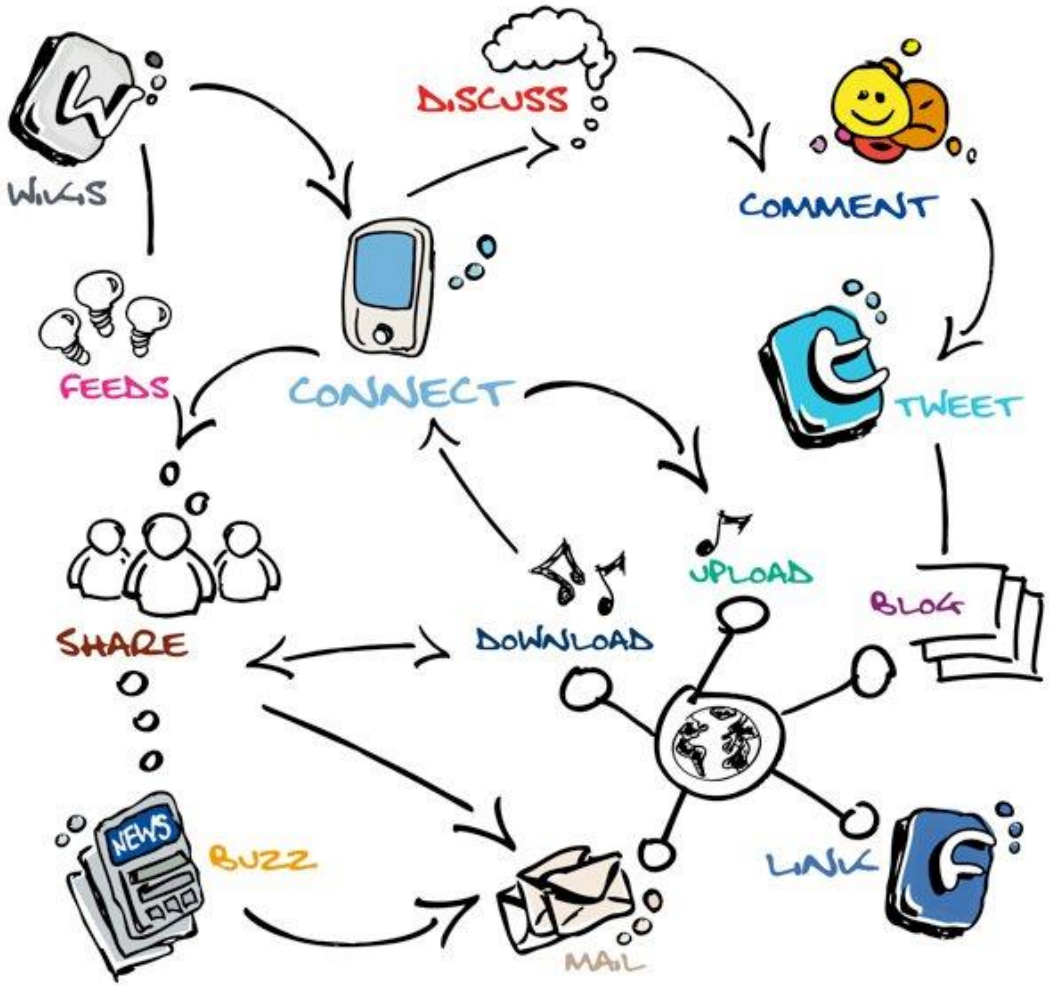
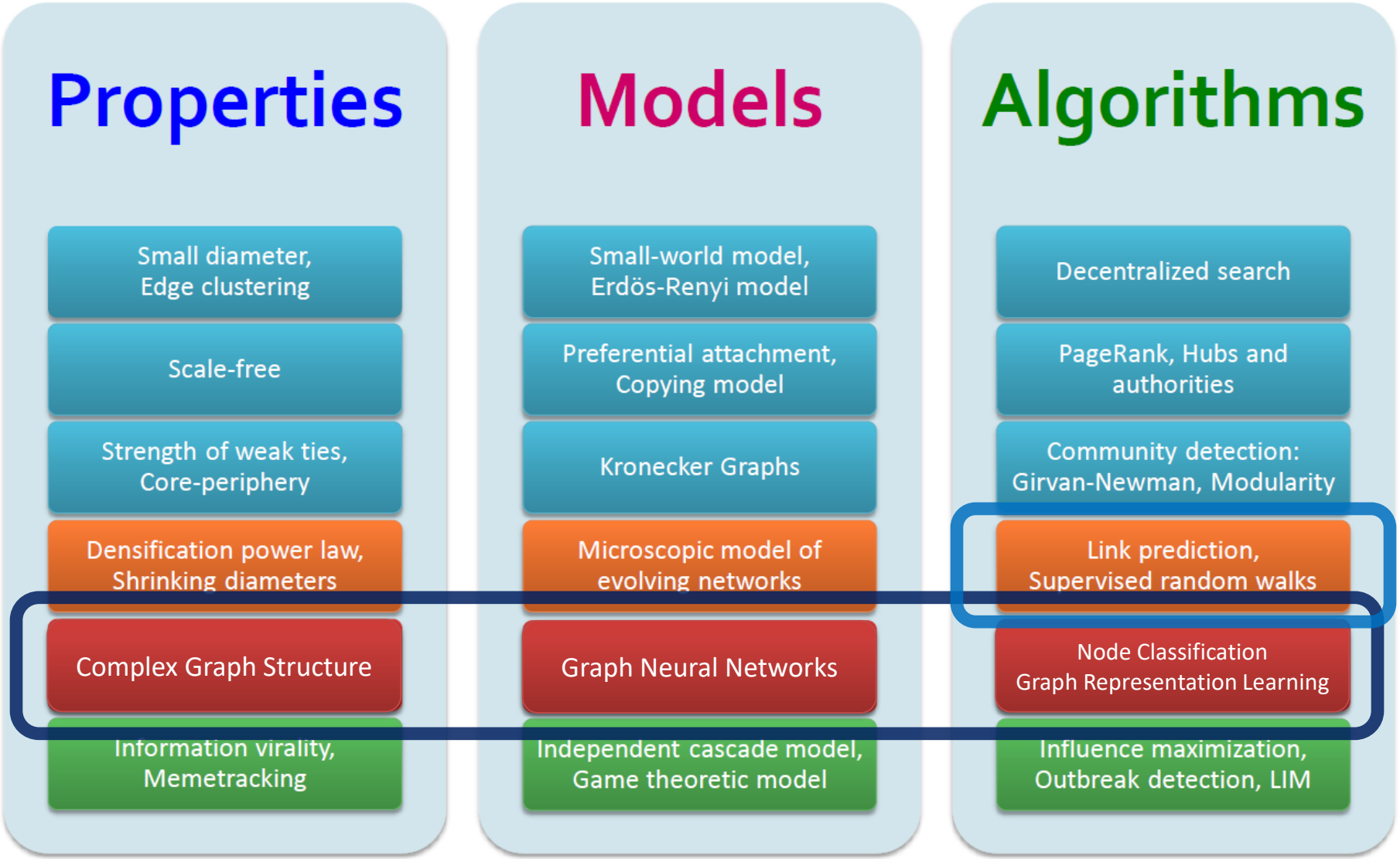


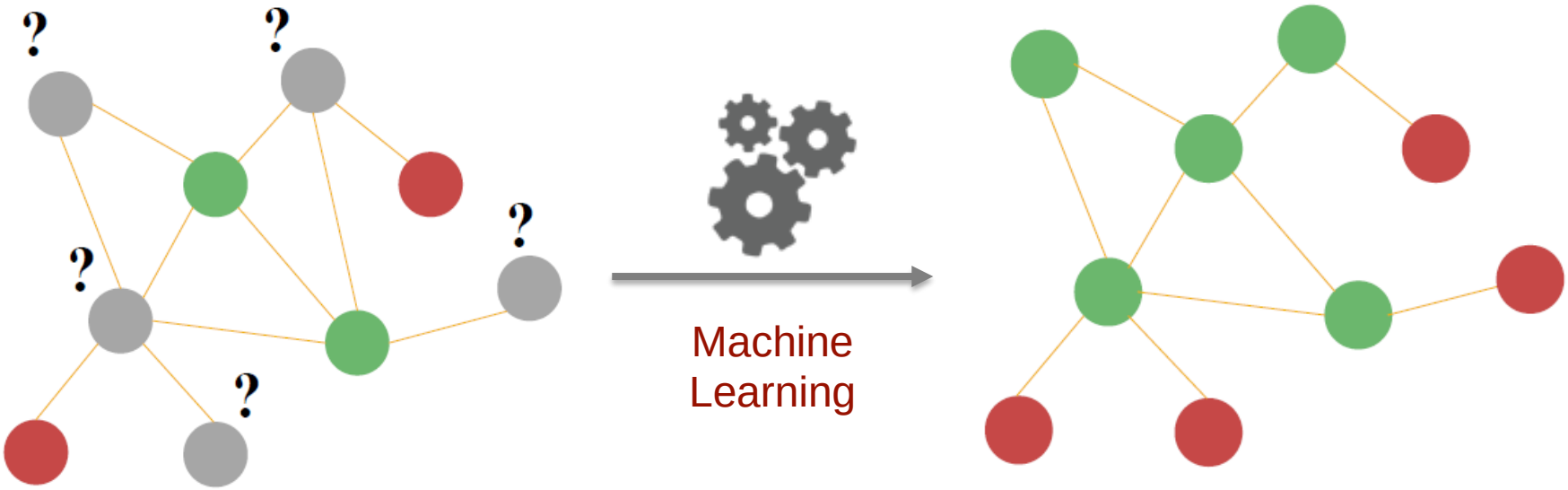
Graph Representation Learning

Ahmet Onur Durahim

How the Class Fits Together

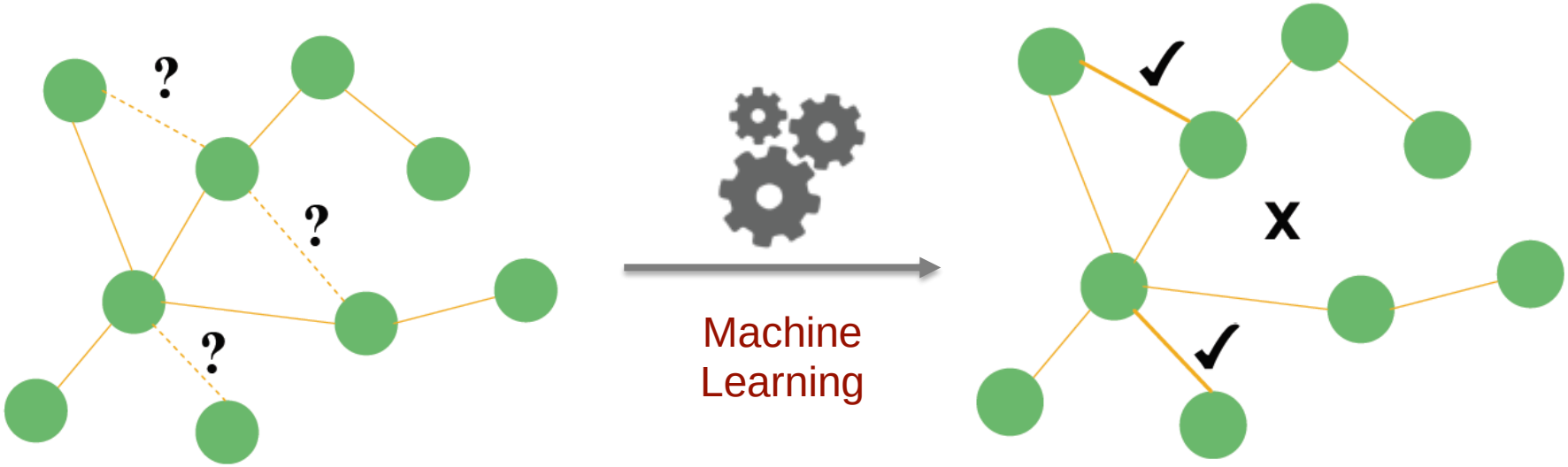


Machine Learning in Networks



Node classification

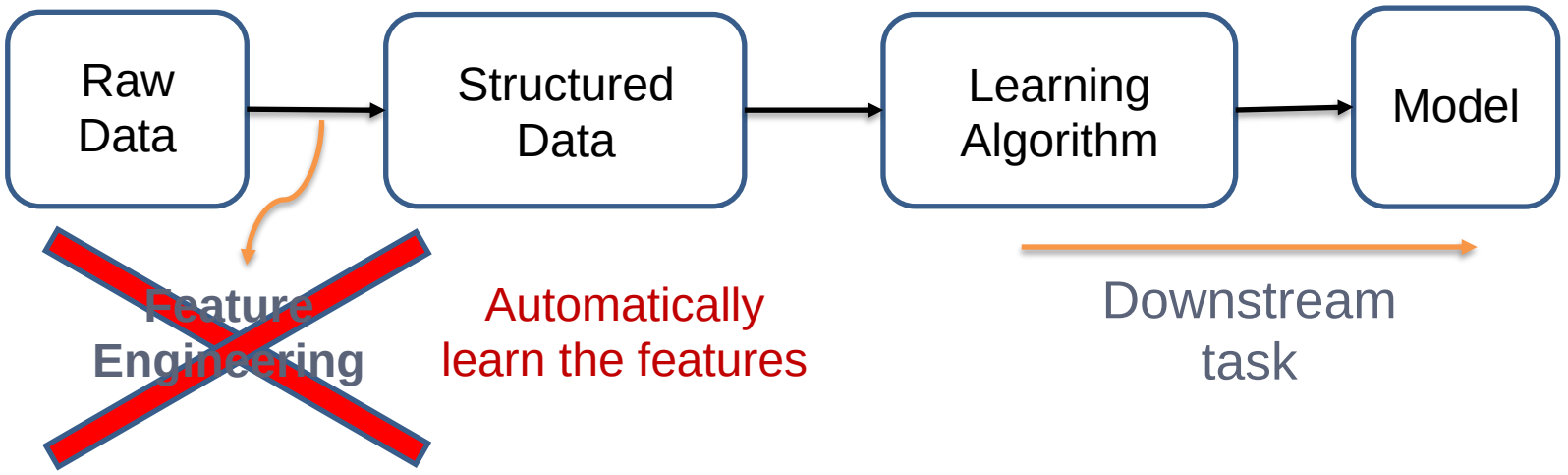
Machine Learning in Networks



Link prediction

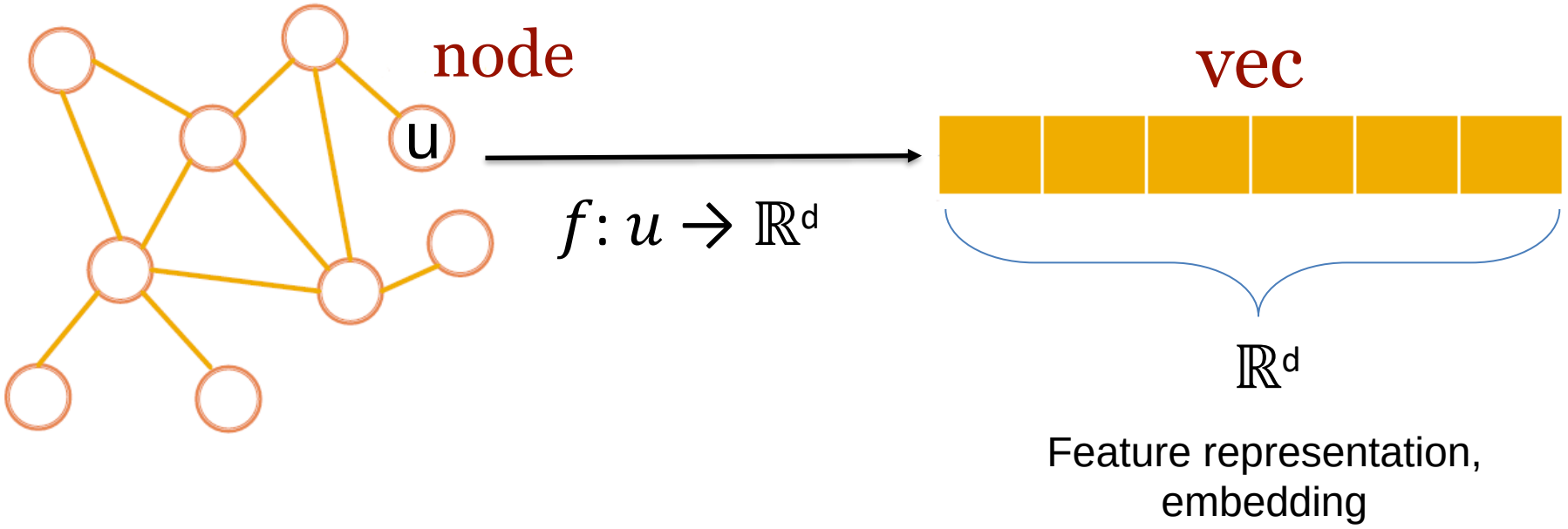
Machine Learning Lifecycle

- ***(Supervised)*** **Machine Learning Lifecycle requires feature engineering every single time!**



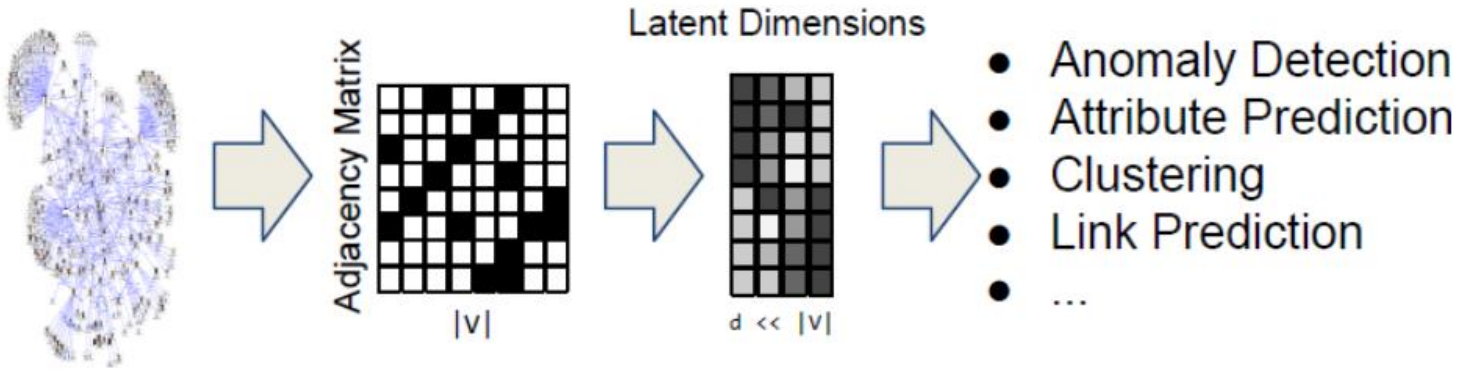
Feature Learning in Graphs

- **Goal:** Efficient ***task-independent feature learning*** for machine learning with graphs!



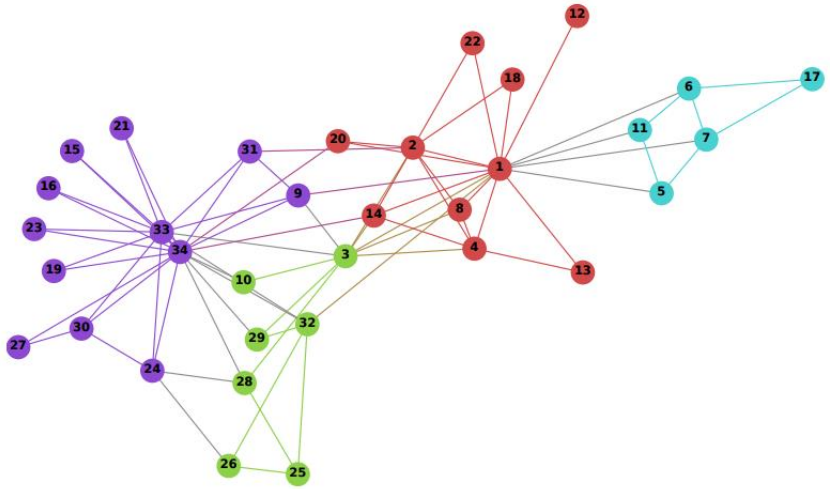
Why network embedding?

- **Task: map each node in a network into a low-dimensional space**
  - Distributed representations for nodes
  - Similarity of embeddings between nodes indicates their network similarity
  - Encode network information and generate node representation

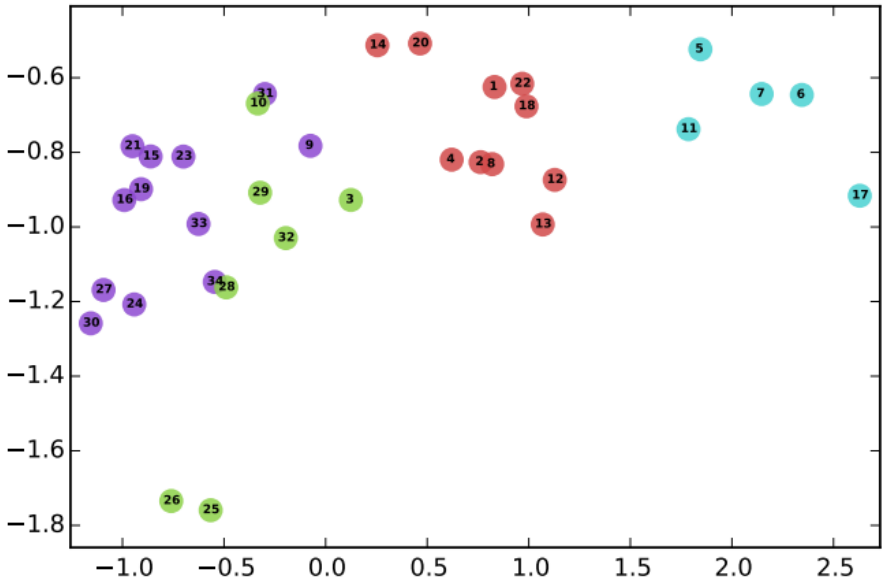


Node Embeddings

- **2D embeddings of nodes of the Zachary's Karate Club network:**



Input: Karate Graph



Output: Representation

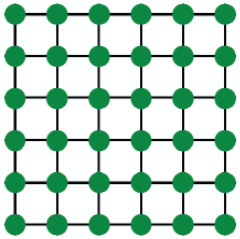
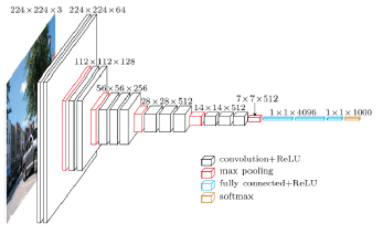
output of DeepWalk method

Vertex colors represent a modularity-based clustering of the input graph.

Image from: Perozzi et al. DeepWalk: Online Learning of Social Representations. *KDD 2014.*

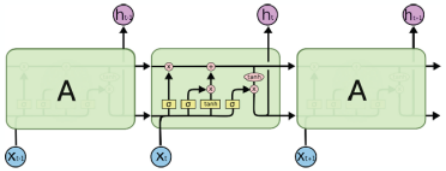
Why Is It Hard?

- Modern deep learning toolbox is designed for simple sequences or grids
  - CNNs for fixed-size images/grids….

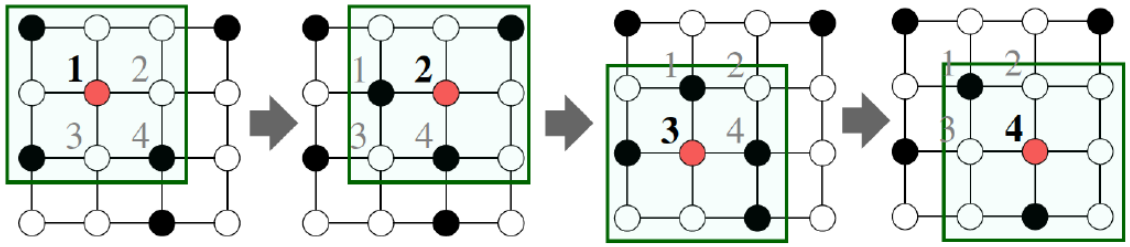
  - RNNs (or word2vec) for text/sequences…





Why Is It Hard?

- But networks are far more complex!
  - Complex topographical structure
    - i.e., no spatial locality like grids



  - No fixed node ordering or reference point
    - i.e., the isomorphism problem
  - Often dynamic and have multimodal features

Embedding Network Nodes

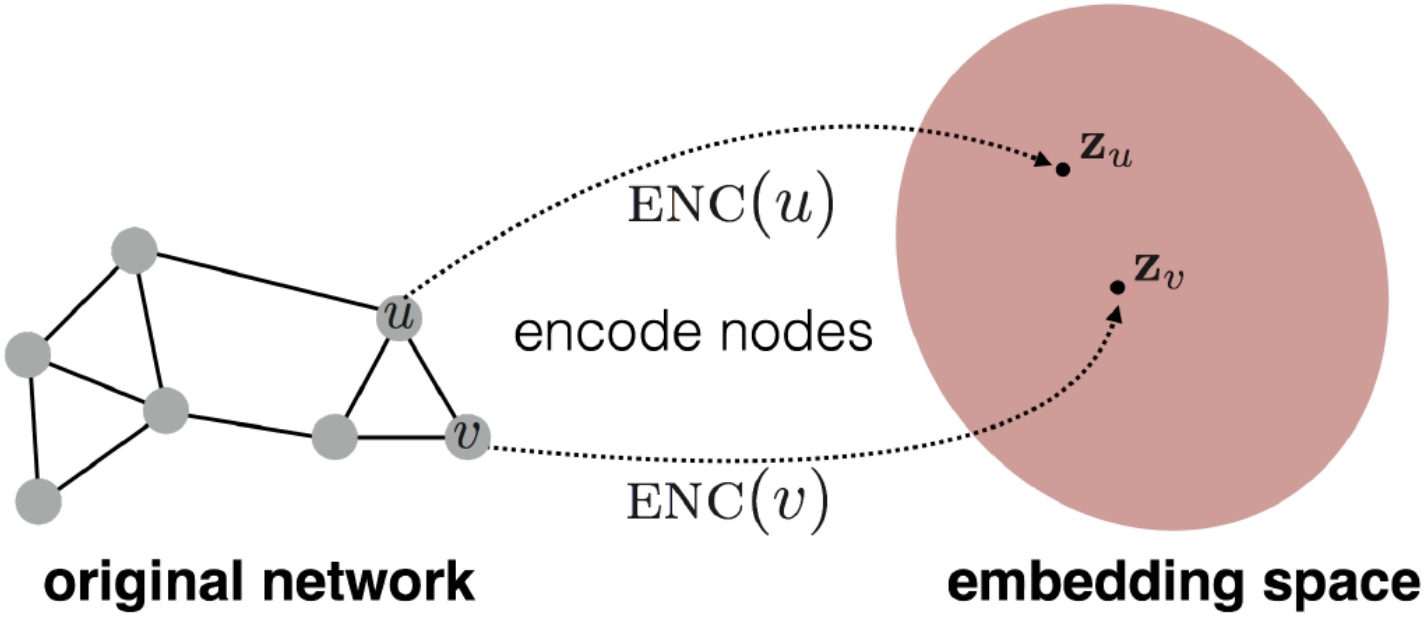
http://web.stanford.edu/class/cs224w/

Setup

- ***Assume we have a graph G:***
  - V is the vertex set
  - A is the adjacency matrix (assume binary)
  - ***No node features or extra information is used!***

Embedding Nodes

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network

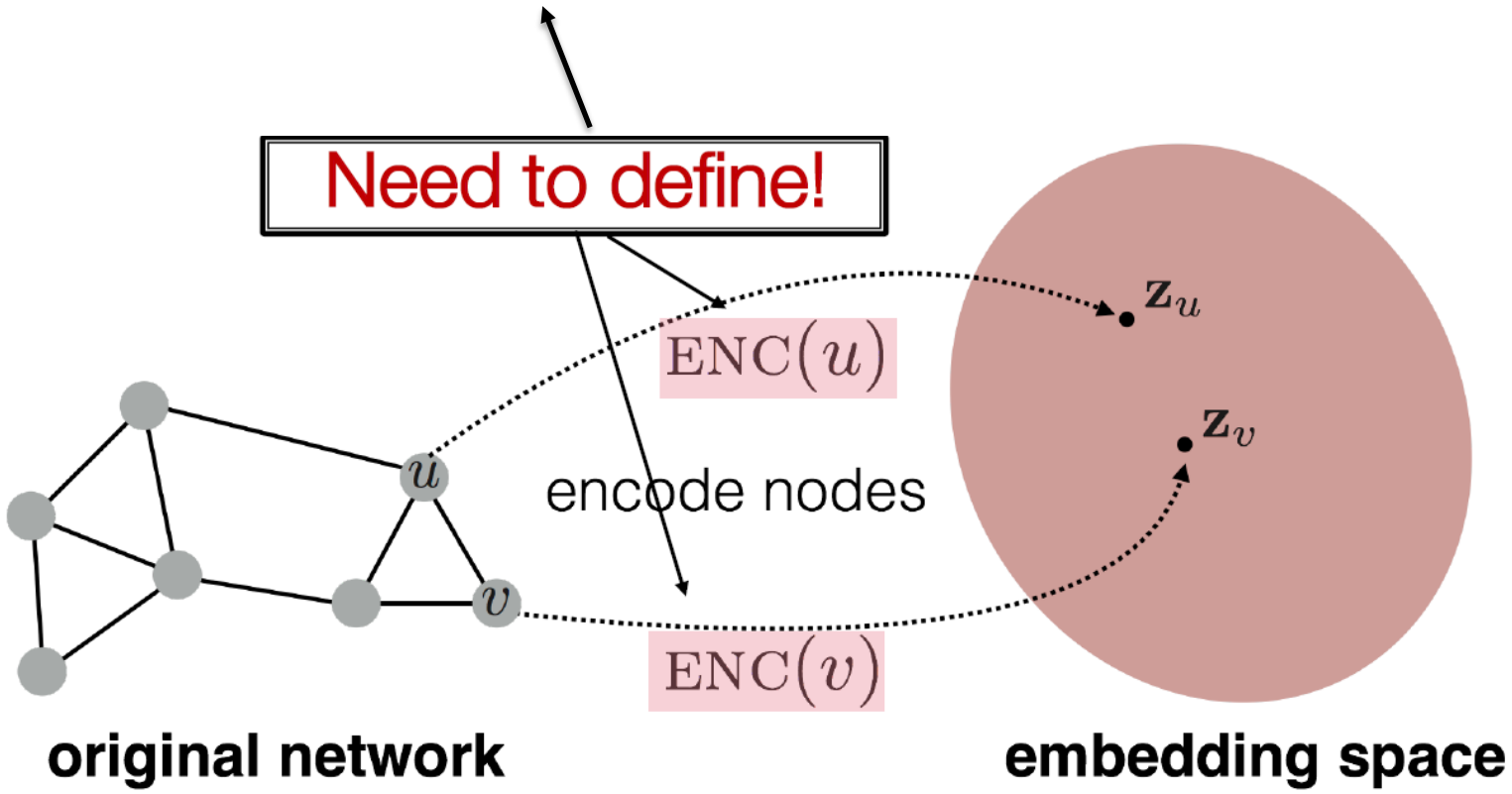


Embedding Nodes

- Goal: $\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$

  similarity(u, v): in the original network

  $\mathbf{z}_v^\top \mathbf{z}_u$: Similarity of the embedding



Learning Node Embeddings

1. ***Define an encoder***
   - i.e., a mapping from nodes to embeddings
2. ***Define a node similarity function***
   - i.e., a measure of similarity in the original network
3. ***Decoder***: maps from embeddings to the similarity score
4. ***Optimize the parameters of the encoder so that:***

$$\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^\top \mathbf{z}_u}_{\text{Similarity of the embedding}}$$

Two Key Components

- ***Encoder:*** maps each node to a low-dimensional vector

$$\text{ENC}(v) = \mathbf{z}_v$$

d-dimensional embedding

node in the input graph

- ***Similarity function:*** specifies how the relationships in vector space map to the relationships in the original network

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Similarity of u and v in the original network

dot product between node embeddings

Shallow Encoding

- ***Simplest encoding approach:*** *encoder is just an embedding-lookup*

$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$$

Matrix:

$\Rightarrow$ each column is a node embedding that is to be learned / optimized

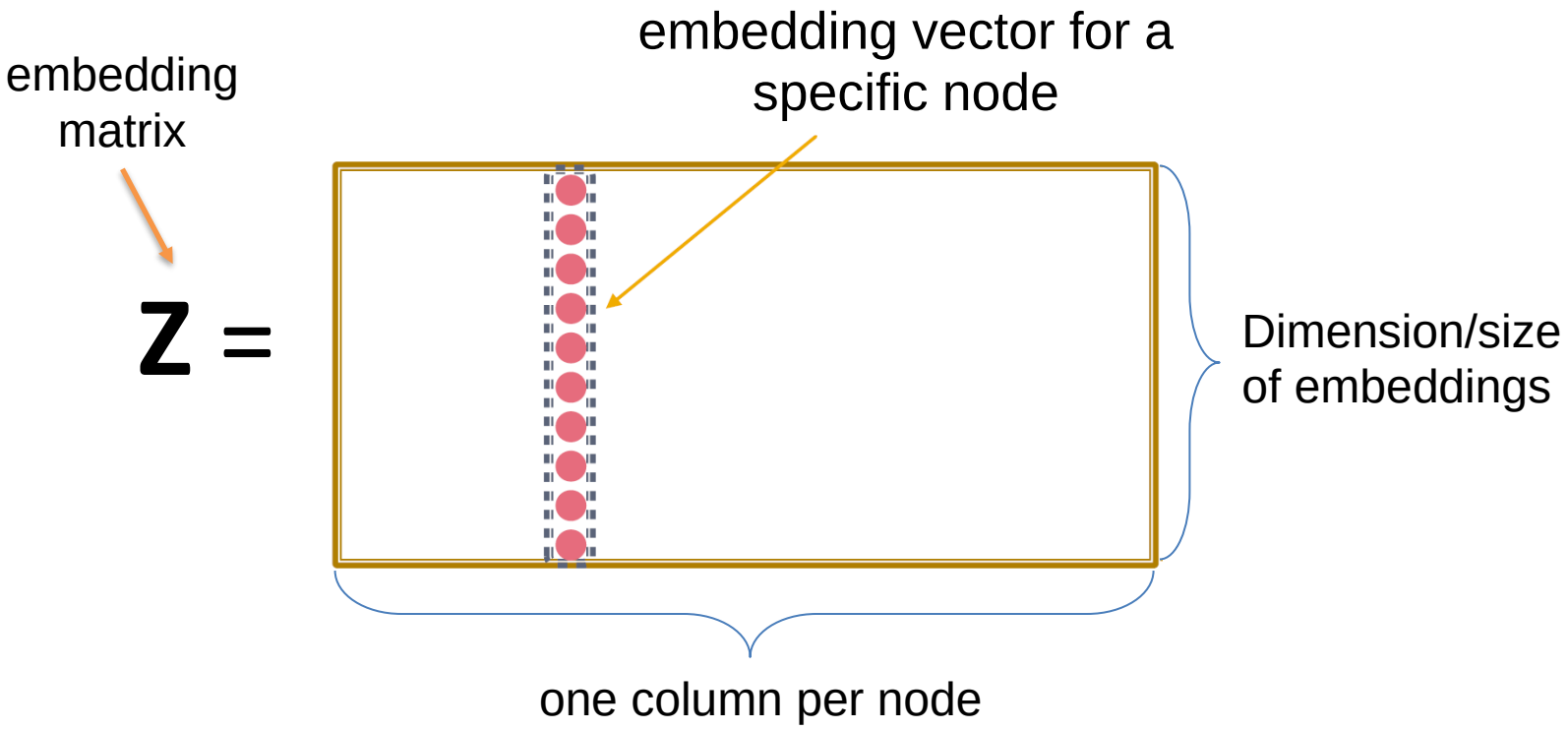
$$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$$

Indicator vector:

$\Rightarrow$ all zeroes except a one in column indicating node v

Shallow Encoding

- ***Simplest encoding approach:*** *encoder is just an embedding-lookup*



Shallow Encoding

- ***Simplest encoding approach:*** *encoder is just an embedding-lookup*

Each node is assigned to a unique embedding vector

(i.e., directly optimize the embedding of each node)

Many methods: *DeepWalk*, LINE, *node2vec*

- ***Decoder: based on node similarity.***
- ***Objective:*** *maximize* $\mathbf{z}_v^\top \mathbf{z}_u$ *for node pairs (u, v) that are similar*

How to Define Node Similarity?

- Key choice of methods is
  - the way they define ***node similarity***
  - should two nodes have similar embeddings if they
    - are connected?
    - share neighbors?
    - have similar "structural roles"?
    - …?

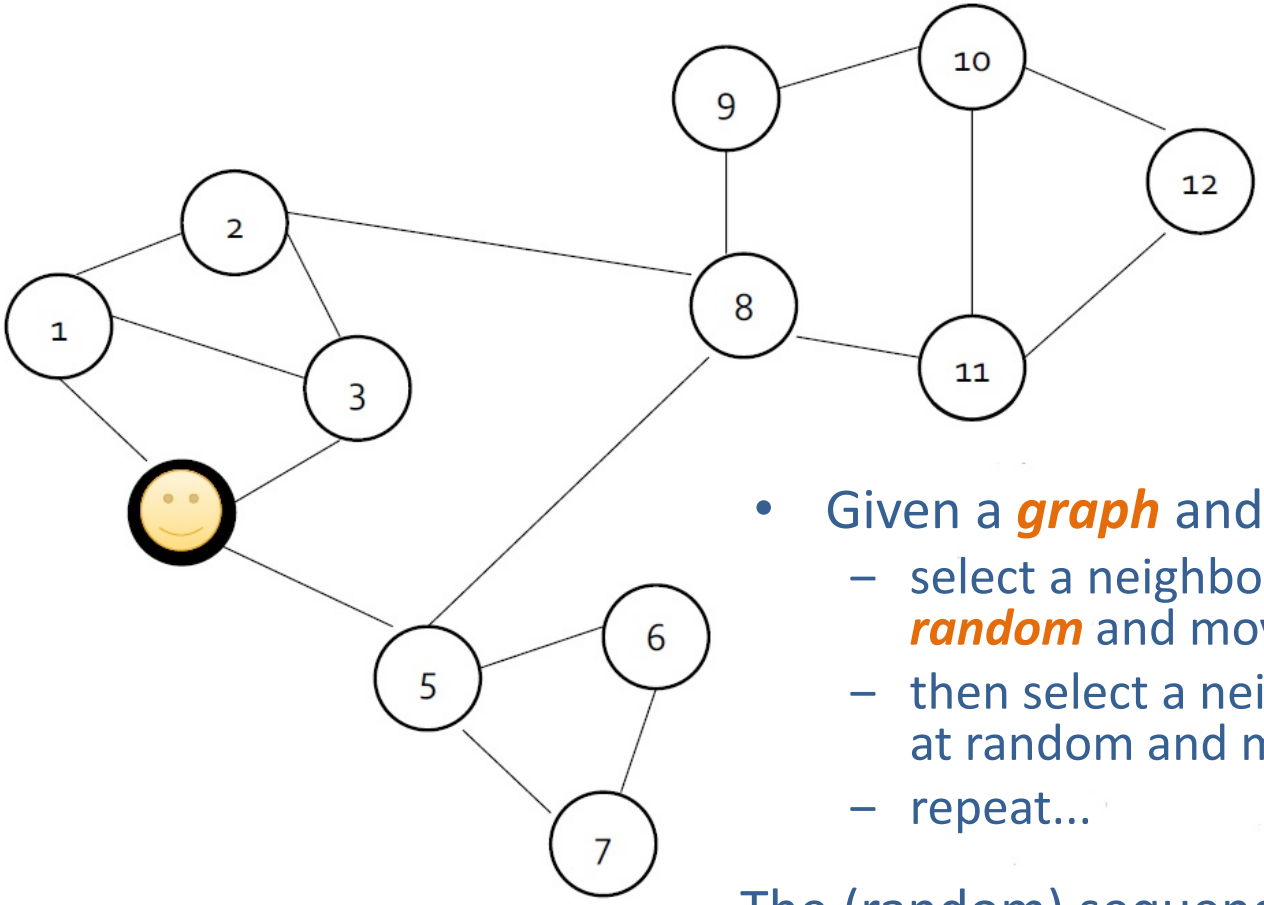
- ***Now:*** discuss how to optimize embeddings for node similarity measure that uses random walks

Random Walk Approaches to Learning Node Embeddings

Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations

Grover et al. 2016. node2vec: Scalable Feature Learning for Networks

Random Walk



- Given a ***graph*** and a ***starting point***
  - select a neighbor of starting point at ***random*** and move to this neighbor
  - then select a neighbor of this point at random and move to it
  - repeat...

The (random) sequence of points selected this way is a ***random walk on the graph***

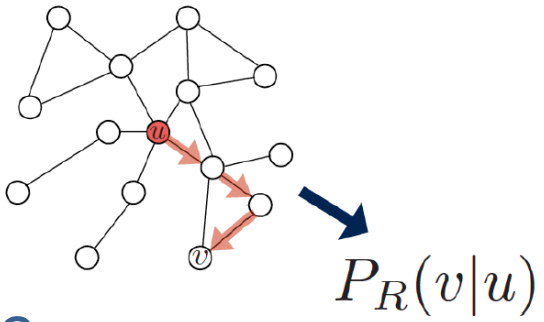
Random-Walk Embeddings

$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$

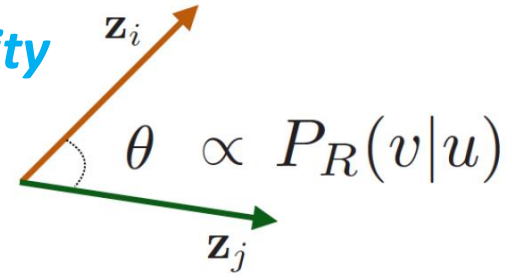
probability that u and v co-occur on a random walk over the network

Random-Walk Embeddings

- Estimate probability of visiting node $\boldsymbol{v}$ on a random walk starting from node $\boldsymbol{u}$ using some random walk strategy $\boldsymbol{R}$



- Optimize embeddings to encode these random walk statistics
  - Similarity encodes ***random walk similarity***
    - similarity: dot product = cos(θ)



Why Random Walks?

- ***Expressivity:*** Flexible stochastic definition of node similarity that incorporates both ***local*** and ***higher-order neighborhood information***
- ***Efficiency:*** do ***not need*** to consider ***all node pairs*** when training
  - need to consider pairs that co-occur on random walks

Unsupervised Feature Learning

- ***Intuition:*** Find embedding of nodes to d-dimensions that preserves similarity
- ***Idea:*** Learn node embedding such that nearby nodes are close together in the network
- Given a node u, ***how do we define nearby nodes***?
  - $\boldsymbol{N_R(u)}$**:** neighbourhood of u obtained by some strategy R

Feature Learning as Optimization

- **Goal:** Given $G = (V, E)$, learn a mapping z: $u \rightarrow \mathbb{R}^d$
- Log-likelihood objective:

$$\max_z \sum_{u \in V} \log \mathrm{P}(N_\mathrm{R}(u) | z_u)$$

  - where $N_R(u)$ is neighborhood of node u by strategy R
- Given node u, learn feature representations that are predictive of the nodes in its neighborhood $N_R(u)$

Random Walk Optimization

- Run short ***fixed-length random walks*** starting from each node on the graph using some strategy *R*
- For each node u collect $N_R(u)$, the multiset* of nodes visited on random walks starting from u
- Optimize embeddings according to: Given node u, predict its neighbors $N_R(u)$

$$\max_{z} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | z_u)$$

*$N_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks

Random Walk Optimization

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

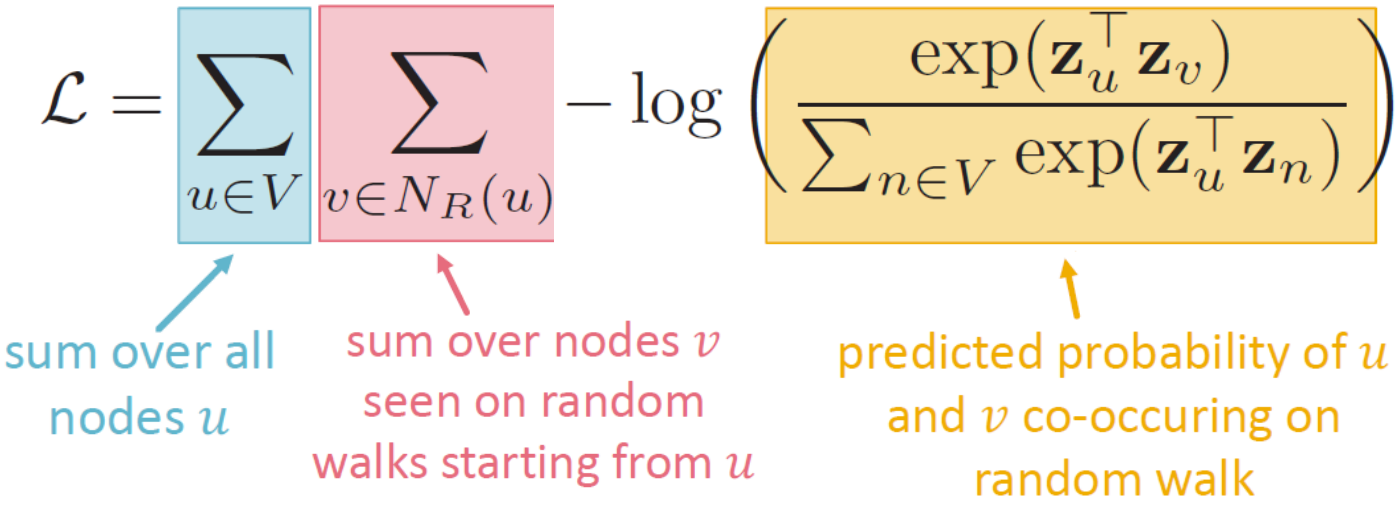
- ***Intuition:*** Optimize embeddings to maximize likelihood of random walk co-occurrences
- ***Parameterize $P(v|\mathbf{z}_u)$ using softmax***

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

Why softmax?
We want node v to be most similar to node u (out of all nodes n).
Intuition: $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

Random Walk Optimization

- **Putting it all together:**



  - Optimizing random walk embeddings = Finding embeddings z_u that minimize L

Random Walk Optimization

- **But doing this naively is too expensive!!**

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

Nested sum over nodes gives
$O(|V|^2)$ complexity!

Random Walk Optimization

- **But doing this naively is too expensive!!**

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

The normalization term from the softmax is the culprit

Can we approximate it?

Negative Sampling

- **Solution:** Negative sampling

Why is the approximation valid?
Technically, this is a different objective. But Negative Sampling is a form of Noise Contrastive Estimation (NCE) which approx. maximizes the log probability of softmax.

New formulation corresponds to using a logistic regression (sigmoid func.) to distinguish the target node v from nodes n_R sampled from background distribution $P_{v.}$

More at https://arxiv.org/pdf/1402.3722.pdf

$$\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n\in V}\exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k}\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

sigmoid function
(makes each term a "probability" between 0 and 1)

random distribution over all nodes

Instead of normalizing w.r.t. all nodes
normalize against k random "**negative samples**" n_i

Negative Sampling

$$\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n\in V}\exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

random distribution over all nodes

$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k}\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

Sample k negative nodes proportional to degree

Two considerations for k (# negative samples)

1. Higher k gives more ***robust estimates***
2. Higher k corresponds to ***higher bias on negative events***

In practice k = 5-20

Random Walk Optimization: Stepping Back

- Run short ***fixed-length random walks*** starting from each node on the graph using some strategy *R*
- For each node u collect $N_R(u)$, the multiset of nodes visited on random walks starting from u
- Optimize embeddings using Stochastic Gradient Descent

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

We can efficiently approximate this using negative sampling!

How should we randomly walk?

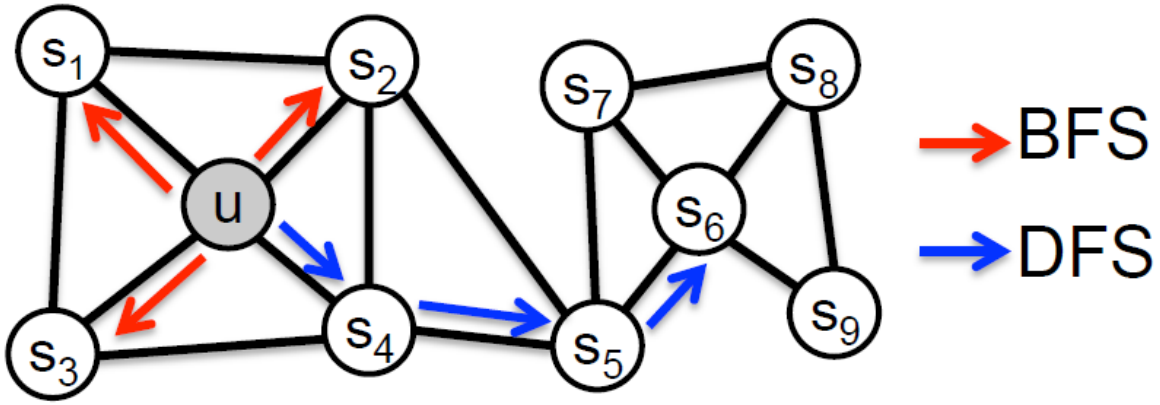
- **Done:** How to optimize embeddings given random walk statistics
- **What strategies should we use to run these random walks?**
  - ***Simplest idea:*** **run fixed-length, unbiased random walks starting from each node**
    - DeepWalk from Perozzi et al., 2013
    - Problem: such notion of similarity is ***too constrained***
  - *How can we generalize this?*

Overview of node2vec

- **Goal:** Embed nodes with similar network neighborhoods close in the feature space
  - framed as a maximum likelihood optimization problem
    - independent to the downstream prediction task
- **Key observation:** Flexible notion of network neighborhood $N_R(u)$ of node u leads to rich node embeddings
  - Develop biased 2nd order random walk R to generate network neighborhood $N_R(u)$ of node u

node2vec: Biased Walks

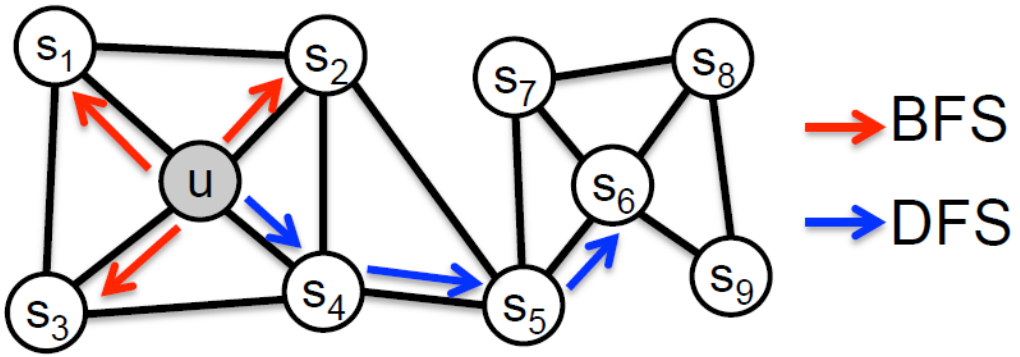
- **Idea:** use flexible, biased random walks that can trade off between **local** and **global** views of the network
  - Grover and Leskovec, 2016



BFS: Breadth First Search
DFS: Depth First Search

node2vec: Biased Walks

- Two classic strategies to define a neighborhood $N_R(u)$ of a given node u:

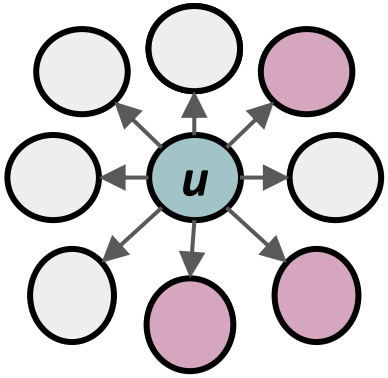


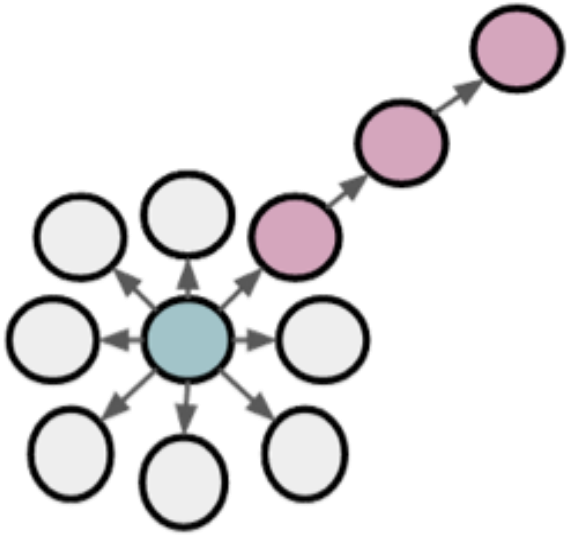
Walk of length 3 ($N_R(u)$ of size 3):

$N_{BFS}(u) = \{s_1, s_2, s_4\}$ Local microscopic view

$N_{DFS}(u) = \{s_4, s_5, s_6\}$ Global macroscopic view

BFS vs. DFS





BFS:
Micro-view of neighbourhood

DFS:
Macro-view of neighbourhood

Interpolating BFS and DFS

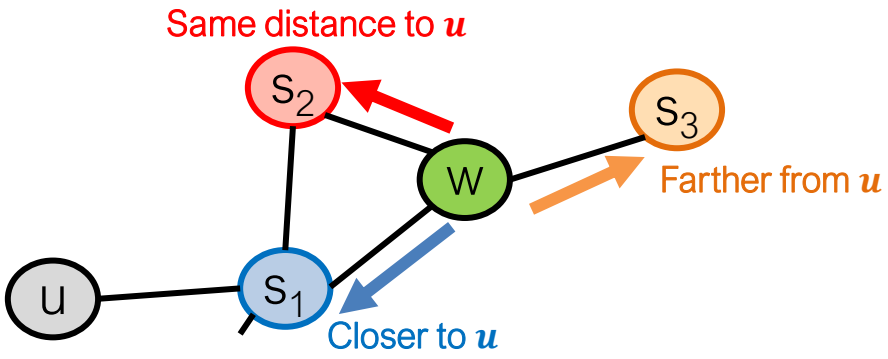
Biased fixed-length random walk $\boldsymbol{R}$ that, given a node $\boldsymbol{u}$, generates neighborhood $\boldsymbol{N_R(u)}$

- Two parameters:
  - **Return parameter $\boldsymbol{p}$:**
    - Return back to the previous node
  - **In-out parameter $\boldsymbol{q}$:**
    - Moving outwards (DFS) vs. inwards (BFS)
    - Intuitively, q is the "ratio" of BFS vs. DFS

Biased Random Walks

Biased 2nd-order random walks explore network neighborhoods:

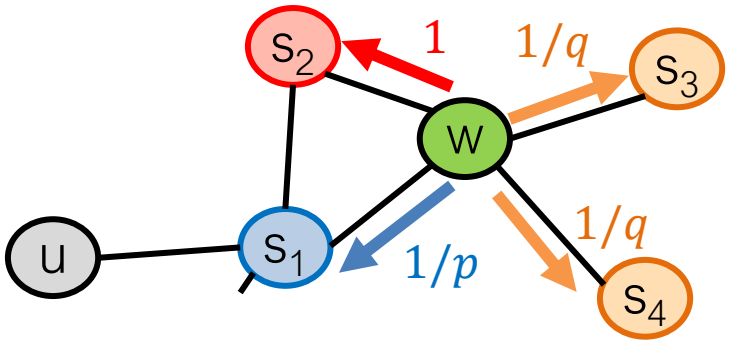
- Random walk just traversed edge (s_1, w) and is now at $\boldsymbol{w}$
- **Insight:** Neighbors of w can only be:



Idea: Remember where that walk came from

Biased Random Walks

- Walker came over edge (s_1,w) and is at ***w***
- Where to go next?

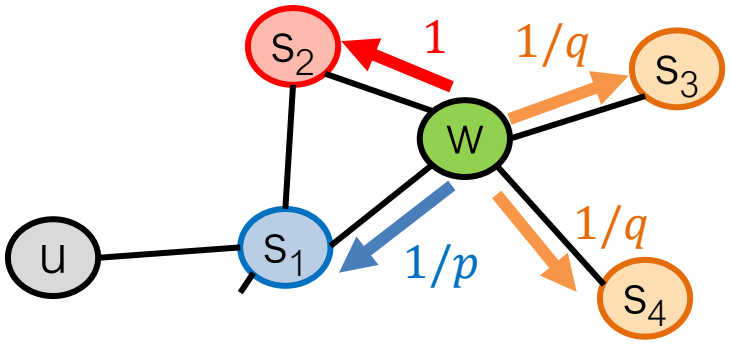


$1/p$, $1/q$, 1 are unnormalized probabilities

- $\boldsymbol{p}$, $\boldsymbol{q}$ model transition probabilities
  - p: return parameter
  - q: "walk away" parameter

Biased Random Walks

- Walker came over edge (s_1,w) and is at ***w***
- Where to go next?



	Target t	Prob.	Dist. (s_1, t)
w →	s_1	$1/p$	0
	s_2	1	1
	s_3	$1/q$	2
	s_4	$1/q$	2

Unnormalized transition prob. segmented based on distance from s_1

BFS-like walk: Low value of p

DFS-like walk: Low value of q

$N_R(u)$ are the nodes visited by the biased walk

node2vec Algorithm

1. Compute random walk probabilities
2. Simulate r random walks of length l starting from each node u
3. Optimize the node2vec objective using Stochastic Gradient Descent

***Linear-time** complexity*

All 3 steps are individually parallelizable

How to Use Embeddings

- **How to use embeddings z_i of nodes:**
  - **Clustering/community detection:** Cluster points z_i
  - **Node classification:** Predict label $f(z_i)$ of node i based on z_i
  - **Link prediction:** Predict edge (i, j) based on $f(z_i, z_j)$
    - concatenate, avg, product, or take a difference between the embeddings:
      - Concatenate: $f(z_i, z_j) = g([z_i, z_j])$
      - Hadamard: $f(z_i, z_j) = g(z_i * z_j)$ (per coordinate product)
      - Sum/Avg: $f(z_i, z_j) = g(z_i + z_j)$
      - Distance: $f(z_i, z_j) = g(||z_i - z_j||_2)$

Summary: Node Embeddings

- **Basic idea:** Embed nodes so that distances in embedding space reflect node similarities in the original network
- **Different notions of node similarity**
  - Adjacency-based (i.e., similar if connected)
  - Multi-hop similarity definitions
  - Random walk approaches **(covered here)**

Summary: Node Embeddings

- **So what method should I use..?**
- No one method wins in all cases….
  - e.g., node2vec performs better on node classification while multi-hop methods performs better on link prediction (Goyal and Ferrara, 2017 survey)
- Random walk approaches are generally more efficient
- **In general:** Must choose definition of node similarity that matches your application!

SPR2EP: A Semi-Supervised Spam Review Detection Framework

An Application of Node Embeddings in Spam Review Detection
SPR2EP, 2018

Motivation - Online Reviews

- Online reviews have great effect on the purchasing decisions of customers
  - Customers need online reviews to learn about the quality of product
  - ***Reliability/Trustworthiness*** of these reviews is an issue in cyberspace
- ***Yelp***, Tripadvisor, Amazon are some of very popular review sites

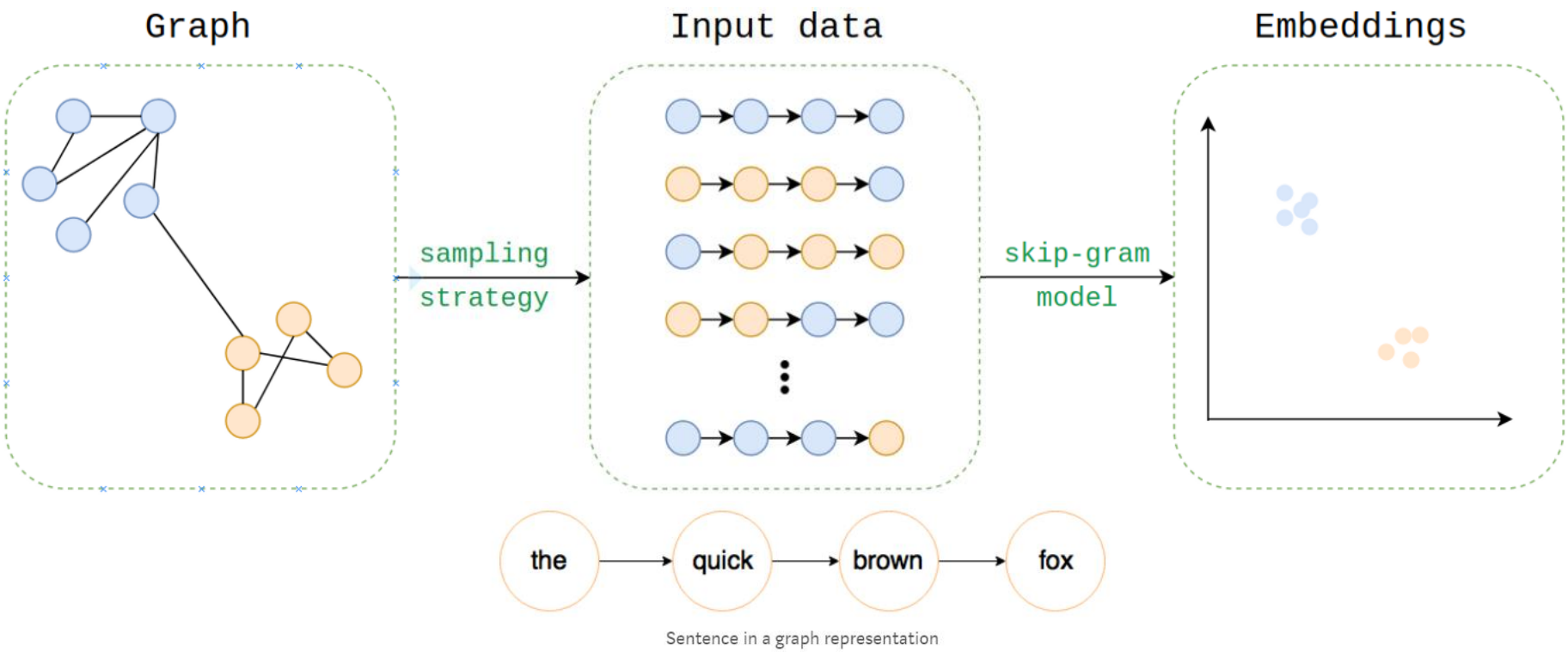
Automatic Spam Review Detection

- ***Review metadata*** is a fruitful source of information which can be utilized in spam review detection
  - ***Product/service specific features***
    - the price and sales rank of the product
    - average review rating of the product
    - the number of reviews written for the product
  - ***Reviewer specific features***
    - the number of reviews written by
    - IP address of the source
    - age of the reviewer accounts
  - ***Network related features***
    - created from the reviews (reviewer/product)
  - Session duration, Behavioral features such as Burstiness, etc.
- This study uses both ***review text*** and ***network information*** for spam review detection

Methodology

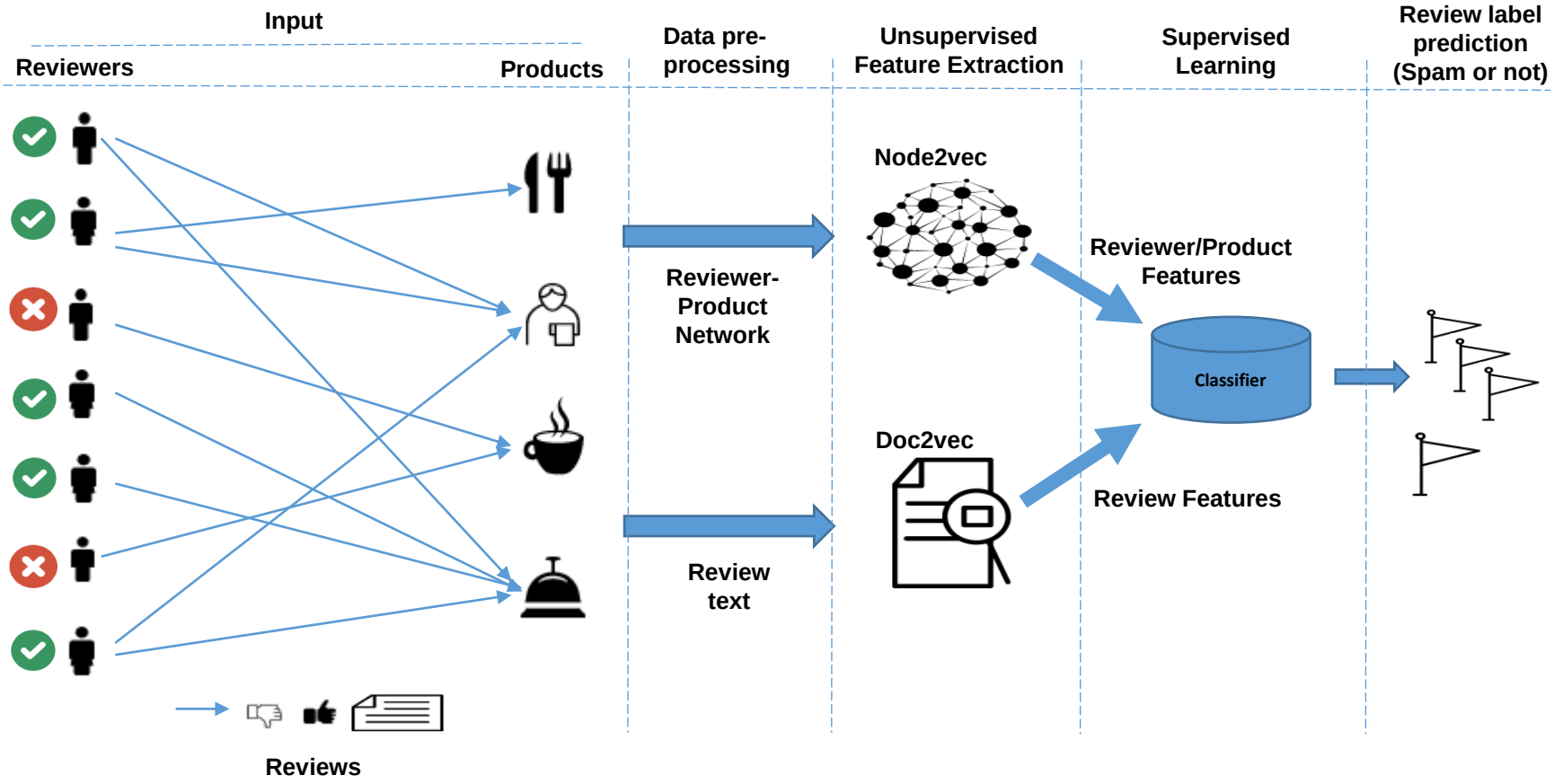
- ***Data Pre-processing***
  - Construct reviewer-product network using review metadata
    - ratings are used to get the weights of the links that connect a review to a product
  - Cleaning of review text
    - removal of the punctuations; making reviews all lowercase; etc.
- ***Feature Generation***
  - Dense low-dimensional vector representations are created
    - ***Doc2vec*** algorithm for reviews, ***Node2vec*** algorithm for nodes
- ***Binary Classification***
  - Dense low-dimensional vector representations are used as input to the Logistic Regression Classifier
  - Logistic Regression Classifier predicts whether a given review is spam or not

Node2vec Embedding Process



https://towardsdatascience.com/node2vec-embeddings-for-graph-data-32a866340fef

Spam Review Detection Framework



Review Datasets

- ***Yelp.com reviews*** for hotels & restaurants in Chicago, and restaurants in New York, and states with NJ, VT, CT, and PA ZIP codes

Dataset	# Reviews (filtered %)	# Users (spammer %)	# Products (rest. & hotel)
YelpChi	67,395 (13.23%)	38,063 (20.33%)	201
YelpNYC	359,052 (10.27%)	160,225 (17.79%)	923
YelpZip	608,598 (13.22%)	260,277 (23.91%)	5,044

- Datasets contains for each review
  - reviewer id, product id, review text, product rating
- Reviews are labeled as ***genuine*** or ***fake*** reviews
  - although the filtering algorithm of YELP could not be perfect, the labels included in these datasets are assumed to be providing us the ground-truth labels

Accuracy Results

	AP			AUC		
	Y'Chi	Y'NYC	Y'Zip	Y'Chi	Y'NYC	Y'Zip
Random	0.1327	0.1028	0.1321	0.5000	0.5000	0.5000
Wang et. al	0.1518	0.1255	0.1803	0.5062	0.5415	0.5982
SpEagle	0.3236	0.246	0.3319	0.7887	0.7695	0.7942
Node2vec	0.3266	0.2518	0.3503	0.7632	0.7648	0.7917
Doc2vec	0.2924	0.2184	0.2838	0.7244	0.7147	0.7213
SPR2EP	**0.3351**	**0.3202**	**0.4220**	**0.8071**	**0.8129**	**0.8318**