Link Prediction Problem

- A fundamental problem in networks
- Here, given a snapshot of a network, need to infer:
  - Which interactions among members is likely
  - Which existing interactions we are missing
- Challenge: Combining information from network with rich node and edge attribute data

How do we combine network structure with node and edge features?

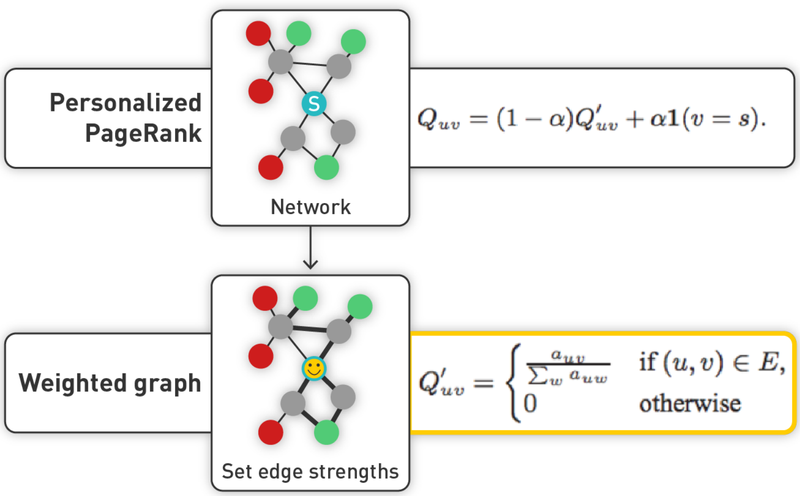
Supervised Link Prediction

- Combination of PageRank with supervised learning
  - **PageRank:** Can capture importance of nodes based on network structure
  - **Supervised learning:** Uses node and edge features to adjust PageRank
- Idea: To "guide" random walk using supervised learning

Supervised Random Walk: Graph + Nodes + Edge Features

- Algorithm developed based on supervised random walks
  - Naturally combines information from the network with node and edge attributes
- Achieved through using attributes to guide a random walk on the graph
- Problem formulated as part supervised machine learning and part random walking with restarts

Friend Graphs: Random Walk With Restarts



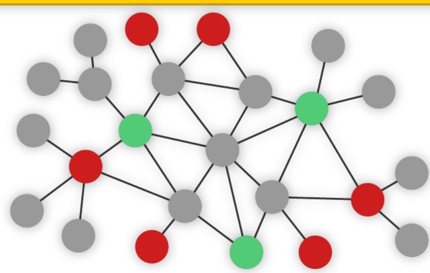
- Set teleportation factor to always teleport back to user.
- Reweight edges using supervised machine learning approach.
- To distribute probability mass, use weighted sum distribution.

Supervised Learning

- Goal: Given a user *s*, recommend friends

Positive: Nodes to which *s* links to in the future

Negative: Nodes to which *s* does not link during this future time period



Social Network Paper

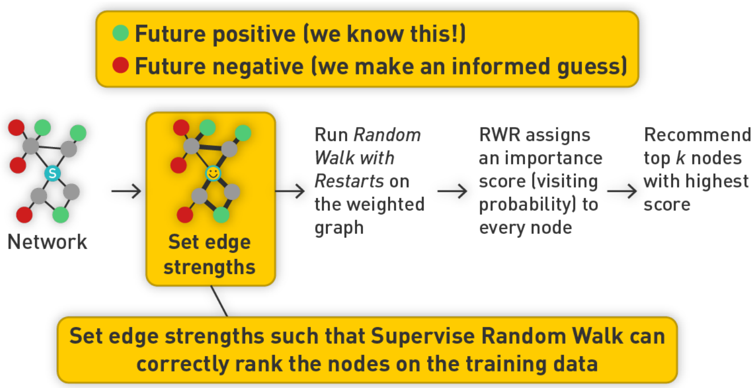
By Lars Backstrom and Jure Leskovic (link)

- Sample: 200 users
  - With a couple hundred friends
  - Active at connecting
- Looked at time sequence of links
  - Positive and negative links selected
- Further partitioned positive and negative examples by time
- Pooled all positive and negative examples for users, put in training and test set

Facebook Study Features

- Seven features generated around each edge (*i, j*)
  - Edge age: T–t–β, where *T* is time cutoff November 1, and *t* is edge creation time. Three features where β=0.10.30.5.
  - Edge initiator: Individual making friend request encoded as +1 or −1
  - Communication and observation features: Probability within a one-week period
  - Common friends: Between *j* and *s*
- All features rescaled to have mean 0 and standard deviation 1; also constant feature of value 1

Hybrid Model



- Random walk with restarts algorithm combined with supervised learning
- Extremely complex, with billions of nodes
- Individual with positive and negative examples to reweight links and social graph

Supervised Random Walk: Process

- Assume *N* individuals (*N* = 100) → 100 social network graphs.
- Weight edges applying weight vector *W* (seven dimensions) to all existing edges.
- Run random walk with restarts (RWR) algorithm (PPR with one node of interest).

$$Q_{uv} = (1-\alpha)Q'_{uv} + \alpha\mathbf{1}(v = s)$$

- Generate steady-state distribution using RWR.
- Teleport to *s* (our node of interest) at every opportunity.

Supervised Random Walk: Process (cont.)

- Steady-state distribution, where each node is scored
- Leads to a list of nodes and corresponding scores
- Take labels and scores and use in training:
  - If node is positive, then positive example
  - If node is negative, then negative example

$$arg\ min_\theta \sum_{p \in P} \sum_{n \in N} \delta(r_p < r_n) + \lambda||\theta||^2$$

Penalty for violating constraint $r_p > r_n$

Positive nodes

Negative nodes

r_x … score of node x on a weighted graph with edge weights $f_\theta(x, y)$

- Ideal: Score for positive examples > Score for negative examples (rp>rn)
- Error: Score for negative examples > Score for positive examples (rprn)

Supervised Portion of Algorithm

Optimization function:

$$\min_{w} F(w) = ||w||^2 + \lambda \sum_{d \in D, l \in L} h(p_l - p_d)$$

- λ trades off between complexity (i.e., norm of *w*) for the fit of model (i.e., how much constraints can be violated).
- Apply gradient descent to objective function.
- Learn a set of weights that can be applied to each edge in our graph.
- Begin another iteration over newly rerated graph.
- Generate new steady-state distribution; revisit problem. Readjust weights if needed.
- Repeat process many times.

Implementation: Two Parts

$$Q'_{uv} = \begin{cases} \frac{a_{uv}}{\sum_w a_{uw}} & \text{if } (u,v) \in E, \\ 0 & \text{otherwise} \end{cases}$$

- Supervised personalized random walk can be done on MapReduce
    - Or using specialized libraries for graphs
- Use gradient descent algorithm on a single server

$$arg\ min_{\theta} \sum \sum_{p \in P} \sum_{n \in N} \delta(r_p < r_n) + \lambda||\theta||^2$$

Positive nodes

Negative nodes

Penalty for violating constraint $r_p > r_n$

r_x … score of node ***x*** on a weighted graph with edge weights $f_\theta(x, y)$