Link Prediction and Network Inference

CS224W: Social and Information Network Analysis
Jure Leskovec, Stanford University
http://cs224w.stanford.edu
Link Prediction in Networks

- **The link prediction task:**
  - Given $G[t_0, t'_0]$ a graph on edges up to time $t'_0$, output a ranked list $L$ of links (not in $G[t_0, t'_0]$) that are predicted to appear in $G[t_1, t'_1]$

- **Evaluation:**
  - $n = |E_{new}|$: # new edges that appear during the test period $[t_1, t'_1]$
  - Take top $n$ elements of $L$ and count correct edges
Link Prediction via Proximity

- **Predict links in a evolving collaboration network**

<table>
<thead>
<tr>
<th></th>
<th>training period</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>authors papers</td>
<td>$</td>
</tr>
<tr>
<td>astro-ph</td>
<td>5343 5816</td>
<td>41852</td>
</tr>
<tr>
<td>cond-mat</td>
<td>5469 6700</td>
<td>19881</td>
</tr>
<tr>
<td>gr-qc</td>
<td>2122 3287</td>
<td>5724</td>
</tr>
<tr>
<td>hep-ph</td>
<td>5414 10254</td>
<td>47806</td>
</tr>
<tr>
<td>hep-th</td>
<td>5241 9498</td>
<td>15842</td>
</tr>
</tbody>
</table>

- **Core:** Because network data is very sparse
  - Consider only nodes with degree of at least 3
    - Because we don't know enough about nodes with less than 3 edges to make good inferences

[LibenNowell-Kleinberg ’03]
Methodology:

- For each pair of nodes \((x, y)\) compute score \(c(x, y)\)
  - For example, \(c(x, y)\) could be the # of common neighbors of \(x\) and \(y\)
- Sort pairs \((x, y)\) by the decreasing score \(c(x, y)\)
  - Note: Only consider/predict edges where both endpoints are in the core (\(\text{deg.} \geq 3\))
- Predict top \(n\) pairs as new links
- See which of these links actually appear in \(G[t_1, t'_1]\)
Different scoring functions $c(x, y) =$
- **Graph distance:** (negated) Shortest path length
- **Common neighbors:** $|\Gamma(x) \cap \Gamma(y)|$
- **Jaccard’s coefficient:** $|\Gamma(x) \cap \Gamma(y)|/|\Gamma(x) \cup \Gamma(y)|$
- **Adamic/Adar:** $\sum_{z \in \Gamma(x) \cap \Gamma(y)} 1/\log |\Gamma(z)|$
- **Preferential attachment:** $|\Gamma(x)| \cdot |\Gamma(y)|$

Then, for a particular choice of $c(\cdot)$
- For every pair of nodes $(x, y)$ compute $c(x, y)$
- Sort pairs $(x, y)$ by the decreasing score $c(x, y)$
- Predict top $n$ pairs as new links
Results: Improvement

Performance score: Fraction of new edges that are guessed correctly.

\[
\sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log |\Gamma(z)|}
\]
Results: Common Neighbors

- Improvement over \#common neighbors
Supervised Random Walks for Link Prediction
Can we learn to predict new friends?

Facebook’s People You May Know

Let’s look at the FB data:
- 92% of new friendships on FB are friend-of-a-friend
- More mutual friends helps
Goal: **Recommend a list of possible friends**

**Supervised machine learning setting:**

- **Labeled training examples:**
  - For every user $s$, have a list of others she will create links to $\{d_1 \ldots d_k\}$ **in the future**
  - Use FB network from May 2012 and $\{d_1 \ldots d_k\}$ are the new friendships you created since then
  - These are the “positive” training examples
  - Use all other users as “negative” example

**Task:**

- For a given node $s$, **score** nodes $\{d_1 \ldots d_k\}$ **higher** than any other node in the network
How to combine node/edge features and the network structure?

- Estimate **strength** of each friendship \((u, v)\) using:
  - Profile of user \(u\), profile of user \(v\)
  - Interaction history of users \(u\) and \(v\)

- This creates a **weighted graph**

- Do **Personalized PageRank from** \(s\) and measure the "**proximity**" (the visiting prob.) of any other node \(w\) from \(s\)

- Sort nodes \(w\) by decreasing "**proximity**"
Supervised Random Walks

- Let $s$ be the starting node
- Let $f_{\beta}(u, v)$ be a function that assigns strength $a_{uv}$ to edge $(u, v)$
  
  $$a_{uv} = f_{\beta}(u, v) = \exp(-\sum_i \beta_i \cdot x_{uv}[i])$$

  - $x_{uv}$ is a feature vector of $(u, v)$
    - Features of node $u$
    - Features of node $v$
    - Features of edge $(u, v)$
  - Note: $\beta$ is the weight vector we will later estimate!

- Do Random Walk with Restarts from $s$ where transitions are according to edge strengths $a_{uv}$
How to estimate edge strengths?
- How to set parameters $\beta$ of $f_\beta(u,v)$?

Idea: Set $\beta$ such that it (correctly) predicts known future links.
Personalized PageRank

- $a_{uv}$ .... Strength of edge $(u, v)$
- Random walk transition matrix:

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

- PageRank transition matrix:

$$Q_{ij} = (1 - \alpha)Q'_{ij} + \alpha 1(j = s)$$

  - Where with prob. $\alpha$ we jump back to node $s$

- Compute PageRank vector: $p = p^T Q$
- Rank nodes $w$ by decreasing $p_w$
Positive examples
\[ D = \{d_1, \ldots, d_k\} \]

Negative examples
\[ L = \{\text{other nodes}\} \]

What do we want?
\[ \min_{\beta} F(\beta) = \|\beta\|^2 \]

such that
\[ \forall \, d \in D, \, l \in L : \, p_l < p_d \]

Note:
- Exact solution to this problem may not exist
- So we make the constraints “soft” (i.e., optional)

We prefer small weights \( \beta \) to prevent overfitting

Every positive example has to have higher PageRank score than every negative example
Want to minimize:

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

- **Loss**: \( h(x) = 0 \) if \( x < 0 \), or \( x^2 \) else

Penalty for violating the constraint that \( p_d > p_l \)
Want to minimize $F(\beta)$

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

Both $p_l$ and $p_d$ depend on $\beta$

- Given $\beta$ assign edge weights $a_{uv} = f_\beta(u, v)$
- Using $Q = [a_{uv}]$ compute PageRank scores $p_\beta$
- Rank nodes by the decreasing score

Goal: Want to find $\beta$ such that $p_l < p_d$
How to minimize $F(\beta)$?

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

Idea:

- Start with some random $\beta^{(0)}$
- Evaluate the derivative of $F(\beta)$ and do a small step in the opposite direction

$$\beta^{(t+1)} = \beta^{(t)} - \eta \frac{\partial F(\beta^{(t)})}{\partial \beta}$$

- Repeat until convergence
Gradient Descent

- What’s the derivative $\frac{\partial F(\beta(t))}{\partial \beta}$?

$$
\frac{\partial F(\beta)}{\partial \beta} = \sum_{l,d} \frac{\partial h(p_l - p_d)}{\partial \beta} + 2\lambda \beta
$$

$$
= \sum_{l,d} \frac{\partial h(p_l - p_d)}{\partial (p_l - p_d)} \left( \frac{\partial p_l}{\partial \beta} - \frac{\partial p_d}{\partial \beta} \right) + 2\lambda \beta
$$

- We know:

$$
p = p^T Q \quad \text{that is} \quad p_u = \sum_j p_j Q_{j,u}
$$

- So:

$$
\frac{\partial p_u}{\partial \beta} = \sum_j Q_{j,u} \frac{\partial p_j}{\partial \beta} + p_j \frac{\partial Q_{j,u}}{\partial \beta}
$$

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

$\lambda(x) = \max\{x, 0\}^2$

Easy!
Gradient Descent

- We just got: 
  \[ \frac{\partial p_u}{\partial \beta} = \sum_j Q_{ju} \frac{\partial p_j}{\partial \beta} + p_j \frac{\partial Q_{ju}}{\partial \beta} \]

  - Few details:
    - Computing \( \frac{\partial Q_{ju}}{\partial \beta} \) is easy. **Remember:** 
      \[ Q'_{uv} = \begin{cases} 
      \frac{a_{uv}}{\sum_w a_{uw}} & \text{if } (u, v) \in E, \\
      0 & \text{otherwise} 
      \end{cases} \]
    
    - We want \( \frac{\partial p_j}{\partial \beta} \) but it appears on both sides of the equation. Notice the whole thing looks like a PageRank equation: \( x = Q \cdot x + z \)

- **As with PageRank we can use the power-iteration to solve it:**
  
  - Start with a random \( \frac{\partial p^{(0)}}{\partial \beta} \)
  
  - Then iterate: 
    \[ \frac{\partial p^{(t+1)}}{\partial \beta} = Q \cdot \frac{\partial p^{(t)}}{\partial \beta} + \frac{\partial Q_{ju}}{\partial \beta} \cdot p \]
To optimize $F(\beta)$, use gradient descent:

- Pick a random starting point $\beta^{(0)}$
- Using current $\beta^{(t)}$ compute edge strengths and the transition matrix $Q$
- Compute PageRank scores $p$
- Compute the gradient with respect to weight vector $\beta^{(t)}$
- Update $\beta^{(t+1)}$
Data: Facebook

- **Facebook Iceland network**
  - 174,000 nodes (55% of population)
  - Avg. degree 168
  - Avg. person added 26 friends/month

- **For every node $s$:**
  - **Positive examples:**
    - $D = \{ \text{new friendships $s$ created in Nov '09} \}$
  - **Negative examples:**
    - $L = \{ \text{other nodes $s$ did not create new links to} \}$
  - **Limit to friends of friends:**
    - On avg. there are 20,000 FoFs (maximum is 2 million)!
Experimental setting

- **Node and Edge features for learning:**
  - **Node:** Age, Gender, Degree
  - **Edge:** Age of an edge, Communication, Profile visits, Co-tagged photos

- **Evaluation:**
  - **Precision at top 20**
    - We produce a list of 20 candidates
      - By taking top 20 nodes $x$ with highest PageRank score $p_x$
    - Measure to what fraction of these nodes $s$ actually links to
Facebook: Predict future friends
- Adamic-Adar already works great
- Supervised Random Walks (SRW) gives slight improvement

<table>
<thead>
<tr>
<th>Learning Method</th>
<th>Prec@Top20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Walk with Restart</td>
<td>6.80</td>
</tr>
<tr>
<td>Adamic-Adar</td>
<td>7.35</td>
</tr>
<tr>
<td>Common Friends</td>
<td>7.35</td>
</tr>
<tr>
<td>Degree</td>
<td>3.25</td>
</tr>
<tr>
<td>SRW: one edge type</td>
<td>6.87</td>
</tr>
<tr>
<td>SRW: multiple edge types</td>
<td>7.57</td>
</tr>
</tbody>
</table>
Results: Facebook

- 2.3x improvement over previous FB-PYMK (People You May Know)

Fraction of Friending from PYMK

Results: Co-Authorship

- **Arxiv Hep-Ph collaboration network:**
  - Poor performance of unsupervised methods
  - SRW gives a boost of 25%!

<table>
<thead>
<tr>
<th>Learning Method</th>
<th>Prec@Top20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Walk with Restart</td>
<td>3.41</td>
</tr>
<tr>
<td>Adamic-Adar</td>
<td>3.13</td>
</tr>
<tr>
<td>Common Friends</td>
<td>3.11</td>
</tr>
<tr>
<td>Degree</td>
<td>3.05</td>
</tr>
<tr>
<td>SRW: one edge type</td>
<td>4.24</td>
</tr>
<tr>
<td>SRW: multiple edge types</td>
<td>4.25</td>
</tr>
</tbody>
</table>
Network Inference

CS224W: Social and Information Network Analysis
Jure Leskovec, Stanford University
http://cs224w.stanford.edu
Many networks are implicit or hard to observe:

- Hidden/hard-to-reach populations:
  - Network of needle sharing between drug injection users
- Implicit connections:
  - Network of information propagation in online news media

But we can observe results of the processes taking place on such (invisible) networks:

- Virus propagation:
  - Drug users get sick, and we observe when they see the doctor
- Information networks:
  - We observe when media sites mention information

Question: Can we infer the hidden networks?
- There is a hidden diffusion network:

- We only see times when nodes get “infected”:
  - Cascade $c_1$: (a,1), (c,2), (b,3), (e,4)
  - Cascade $c_2$: (c,1), (a,4), (b,5), (d,6)

- Want to infer who-infests-whom network!
Examples and Applications

- Information diffuses through the blogosphere

- We only see the mention but not the source
- Can we reconstruct (hidden) diffusion network?
Examples and Applications

**Virus propagation**
- Process: Viruses propagate through the network.
- We observe: We only observe when people get sick.
- It’s hidden: But NOT who infected whom.

**Word of mouth & Viral marketing**
- Process: Recommendations and influence propagate.
- We observe: We only observe when people buy products.
- It’s hidden: But NOT who influenced whom.

Can we infer the underlying network?
Inferring the Diffusion Network

Network $G^*$

Cascade $c_1$

Cascade $c_2$

Cascade $c_3$

Node $i$
Goal: Find a graph G that best explains the observed infection times

- Given a graph G, define the likelihood $P(C|G)$:

$P_c(a,b)$: How likely is $a$ to infect $b$

$P(c|T)$: How likely is $c$ to propagate via cascade-tree $T$

Here: $T=\{a \rightarrow b \rightarrow c\}$

$P(c|G)$: How likely is $c$ to propagate in graph $G$

In both $T_1, T_2$ the order of infections is the same: $a, b, c$
Goal: Find a graph $G$ that best explains the observed infection times

- Given a graph $G$, define the likelihood $P(C|G)$:
  - Define a model of information diffusion over a graph
  - $P_c(a,b)$ ... prob. that $a$ infects $b$ in contagion $c$
  - $P(c|T)$ ... prob. that $c$ spread in particular cascade-tree $T$
  - $P(c|G)$ ... prob. that cascade $c$ occurred in $G$
  - $P(C|G)$ ... prob. that a set of cascades $C$ occurred in $G$

Questions:
- How to efficiently compute $P(G|C)$? (given a single $G$)
- How to efficiently find $G^*$ that maximizes $P(G|C)$? (over $O(2^{N*N})$ graphs)
Continuous time cascade diffusion model:

- Cascade $c$ reaches node $u$ at $t_u$ and spreads to $u$’s neighbors:
  - With probability $\beta$ cascade propagates along edge $(u, v)$ and we determine the infection time of node $v$
  
  $$t_v = t_u + \Delta$$
  
  e.g.: $\Delta \sim \text{Exponential}$

We assume each node $v$ has only one parent!
The model for one cascade:

- Cascade reaches node $u$ at time $t_u$, and spreads to $u$’s neighbors $v$:
  - With prob. $\beta$ cascade propagates along edge $(u,v)$ and $t_v = t_u + \Delta$

Transmission probability:

$$P_c(u,v) \propto P(t_v - t_u) \text{ if } t_v > t_u \text{ else } \varepsilon$$

- $\varepsilon$ captures influence external to the network
  - At any time a node can get infected from outside with small probability $\varepsilon$, equal for all nodes
Cascade Probability

- **Given node infection times & cascade-tree \( T \):**
  - \( c = \{ (a,1), (c,2), (b,3), (e,4) \} \)
  - \( T = \{ a \rightarrow b, a \rightarrow c, b \rightarrow e \} \)

- **Prob. that \( c \) propagates in cascade-tree \( T \)**

\[
P(c|T) = \prod_{(u,v) \in E_T} \beta P_c(u,v) \prod_{u \in V_T, (u,v) \in E \setminus E_T} (1 - \beta)
\]

Edges that “propagated” \( E_T \)

Edges that failed to “propagate” \( E \setminus E_T \)

- **Approximate it as:**

\[
P(c|T) \approx \prod_{(u,v) \in E_T} P_c(v,u)
\]
Complication: Too Many Trees

- How likely is cascade $c$ to spread in graph $G$?
  - $c = \{(a,1), (c,2), (b,3), (e,4)\}$

- Need to consider all possible ways for $c$ to spread over $G$ (i.e., all spanning trees $T$):

$$P(c|G) = \sum_{T \in \mathcal{T}_c(G)} P(c|T) \approx \max_{T \in \mathcal{T}_c(G)} P(c|T)$$

Consider only the most likely propagation tree
The Optimization Problem

- Score of a graph $G$ for a set of cascades $C$:
  \[
  P(C|G) = \prod P(c|G) \\
  F_C(G) = \sum_{c \in C} \log P(c|G)
  \]

- Want to find the “best” graph:
  \[
  G^* = \arg\max_{|G| \leq k} F_C(G)
  \]

The problem is **NP-hard**: MAX-$k$-COVER [KDD ’10]
Given a cascade $c$, what is the most likely propagation tree?

$$\max_{T \in \mathcal{T}_c(G)} P(c|T) = \max_{T \in \mathcal{T}(G)} \sum_{(i,j) \in T} w_c(i,j)$$

- **Maximum directed spanning tree**
  - Edge $(i,j)$ in $G$ has weight $w_c(i,j) = \log P_c(i,j)$
  - The maximum weight spanning tree on infected nodes: Each node picks an in-edge of max weight:

$$= \sum_{i \in V} \max_{Par_T(i)} w(Par_T(i), i)$$

Local greedy selection gives optimal tree!
Theorem:

$F_c(G)$ is monotonic, and submodular

Proof:

- Single cascade $c$, some edge $e=(r,s)$ of weight. $w_{rs}$
- Show $F_c(G \cup \{e\}) - F_c(G) \geq F_c(G' \cup \{e\}) - F_c(G')$
- Let $w_s$ be max weight in-edge of $s$ in $G$
- Let $w'_s$ be max weight in-edge of $s$ in $G'$
- Since $G \subseteq G'$: $w_s \leq w'_s$ and $w_{rs} = w'_rs$
- $F_c(G \cup \{(r,s)\}) - F_c(G)$
  \[= \max(w_s, w_{rs}) - w_s\]
  \[\geq \max(w'_s, w_{rs}) - w'_s\]
  \[= F_c(G' \cup \{(r,s)\}) - F_c(G')\]
The NetInf algorithm:

Use **greedy hill-climbing** to maximize $F_C(G)$:

- Start with empty $G_0$ ($G$ with no edges)
- Add $k$ edges ($k$ is parameter)
- At every step $i$ add an edge to the graph $G_i$ that maximizes the marginal improvement

\[
e_i = \underset{e \in G \setminus G_{i-1}}{\text{argmax}} \ F_C(G_{i-1} \cup \{e\}) - F_C(G_{i-1})
\]

Note: This is the same algorithm we used for influence maximization
Experiments: Synthetic data

- **Synthetic data:**
  - Take a graph $G$ on $k$ edges
  - Simulate info. diffusion
  - Record node infection times
  - Reconstruct $G$

- **Evaluation:**
  - How many edges of $G$ can NetInf find?
    - Break-even point (precision=recall): 0.95
    - Performance is independent of the structure of $G$!
NetInf achieves \( \approx 90\% \) of the best possible network!

![Graph showing the value of the objective function against the number of edges, comparing NetInf and the upper bound (Th. 4).](image-url)
How Many Cascades Do We Need?

- With $2x$ as many infections as edges, the break-even point is already $0.8 - 0.9$!
Experiments: Real data

- **Memetracker dataset:**
  - 172m news articles
  - Aug ‘08 – Sept ‘09
  - 343m textual phrases
  - Times $t_c(w)$ when site $w$ mentions phrase $c$

- **Given times when sites mention phrases**
- **Infer the network of information diffusion:**
  - Who tends to copy (repeat after) whom

[http://memetracker.org](http://memetracker.org)

Example: Diffusion Network

- 5,000 news sites:

Blogs
Mainstream media
Diffusion Network (small part)

Blogs

Mainstream media