Ranking von Knoten

Ziel: Selektieren und Ranken besonders relevanter und interessanter Knoten/Subgraphen in großen Netzwerken.

Aufgaben:

- Ranking der Knoten nach Einfluss
- Suche nach Schlüsselknoten

Anwendungen:

- Ranking von Webpages
- Bestimmung von zentralen Personen in sozialen und Zitationnetzwerken
- Ranking von Kreuzungen in Straßennetzwerken

Zentralität eine Knotens im Netzwerk

Idee: Zentralität hängt von der Position des Knotens bzgl. der Netzwerkdistanz (= Kosten des kostenminimalen Pfades) ab.

Sei d(v,t) die Länge des kürzesten Pfades von v zu t (v,t $\in$V) in G(V,E):

- **Closeness Centrality:** $C_C(v) = \frac{1}{\sum_{t \in V} d(v,t)}$
- **Graph Centrality:** $C_G(v) = \frac{1}{\max_{t \in V}(d(v,t))}$

Sei σ_{st} Anzahl der kürzesten Pfade von s nach t und sei $\sigma_{st}(v)$ Anzahl kürzeste Pfade von s nach t, die v enthalten.

- **Stress Centrality:** $C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$
- **Betweeness Centrality:** $C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$

Zentralität eine Knotens im Netzwerk

Beispiel: Betrachte Knoten als Router

Bei Ausfall des Routers mit der höchsten Betweenness Centrality, fallen am meisten direkte Kommunikationsverbindungen aus.

Berechnung: Menge aller kürzesten Pfade lässt sich mit dem Algorithmus von Floyd-Warshal in $O(n^3)$ bzgl. und in $O(n^2)$ *bzgl.* Speicher bestimmen.

Lemma: v liegt genau dann auf einem kürzesten Pfad von s nach t wenn gilt

$d(s,t) = d(s,v) + d(v,t)$

$$\Rightarrow \sigma_{st}(v) = \begin{cases} 0 & wenn \quad d(s,t) < d(s,v) + d(v,t) \\ \sigma_{sv} \cdot \sigma_{vt} & sonst \end{cases}$$

$\Rightarrow$ zur Berechnung der Betweeness sind nicht alle Pfade notwendig

$\Rightarrow$ *Es existieren schnellere Lösungen:*

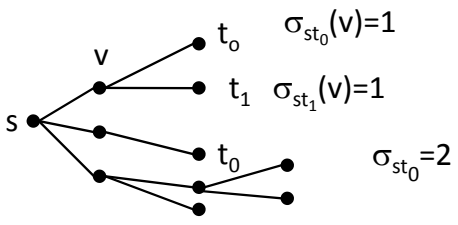
- $O(nm)$ ohne Kantengewichte
- $O(nm+n^2 \log n)$ mit Kantengewichten

mit $n = |V|$ und $m = |E|$ für den Graph $G(V,E)$

Berechnung der Betweeness Centrality

Grundidee:

- Starte für jeden Knoten s einen best-first Suche zu allen anderen Knoten. Ergebnis ist ein Baum der alle kürzesten Pfade in s zu allen anderen Knoten t in G enthält. (wird auch als Dijkstra-Baum bezeichnet)
- Laufe von jedem Blatt zurück in Richtung s und zähle die Knoten, die im Baum hinter jedem Knoten v liegen ($\sigma_{st}(v)$) und die Menge der kürzesten Pfade (σ_{st})



Algorithmus für ungewichtet Graphen(1)

Erläuterungen:

- S: Stack der die Knoten nach Distanz zu Startknoten s aufsammelt
- Q: Priority Queue die Knoten nach Distanz zu s verwaltet (best-first search)
- P[v]: Liste mit Vorgängern von v auf kürzesten Pfaden
- d[v]: aktuell kürzester Pfad von s zu v
- σ[v]: Anzahl der kürzesten Pfade von s zu v
- δ[v]: Sei $\delta_{st}[v] = \frac{\sigma_{st}[v]}{\sigma_{st}}$ dann ist $\delta[v] = \delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}[v] = \sum_{w:v \in P[w]} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w))$

Ablauf:

1. Phase: Algorithmus berechnet Dijkstra-Baum bzgl. aktuellem s
2. Phase: Stack S wird abgebaut und dabei wird die Anzahl der hinter dem Knoten Ziele nach vorne weitergegeben.

Algorithmus für ungewichtet Graphen(2)

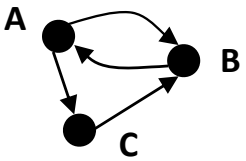
```
CB[v] := 0 ∀ v∈V
for s ∈ V
  S:= empty Stack;
  P[w] := empty List ∀ w∈V;
  σ[t] :=0  ∀t∈V; σ[s]:=1;
  d[t] :=-1 ∀t∈V; d[s]:0;
  Q := empty Queue;
  Q.push(0,s);
  while Q not empty do
    v := Q.pop();
    S.push(v);
    foreach neighbor w of v do
      if d[w] < 0 then
        d[w]:=d[v]+1;
        Q.push(d[w],w);
      end if
      if d[w]=d[v]+1 then
        σ(w):=σ(w)+σ(v)
        P[w].add(v)
      end if
    end for
  end while
  δ[v]:=0; v∈V;
  while S not empty do
    w:=S.pop();
    for v∈P[w] do
      δ[v]:= δ[v]+σ[v]/σ[w] · (1+δ[w]);
    end for
    if w≠s then
      CB[w]:=CB[w]+δ[w];
    end if
  end while
end for
```

PageRank: (S.Brin/B. Page 1996)

- wichtige Kompente im Ranking moderner Suchmaschinen.
  (in Kombination mit Ankertexten, Abstand, Worthäufigkeit)
- Web als stark verbundener, gerichteter Graph G(V,E).
  (Betrachtet gesamten Graph aus bekannten Webpages
  z.B. alle in Suchmaschine gespeicherten Pages.)
- Zufallssurfer macht unendlichen Random-Walk.
  Suche: Aufenthaltswahrscheinlichkeit für Page v
  = „Prestige“ der Webpage v.

Berechnung Pagerank
Startverteilung: *p0(u) = 1 / |V |*

$$E = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$



Adjazenzmatrix E

Übergangsw‘keit: $L[u,v] = \frac{E[u,v]}{\sum_{\beta} E[u,\beta]}$

$$L = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

W‘keit für Zeitpunkt i und Page v: $p_i[v] = \sum_{u \in V} L[u,v] p_{i-1}(u)$

Als Vektor aller Knoten: $\vec{p}_i = L^T \vec{p}_{i-1}$

Berechnung über „Power Iterations“: $\vec{p}_i \leftarrow L^T \vec{p}_{i-1}$

nach ca. 20-30 Iterationen stabil

Lösung für nicht stark verbundene Graphen: 1. Weglassen von Knoten ohne Links
2. Zufallsprünge auf beliebige Seite

HITS (Kleinberg 1998): Hyperlink Induced Topic Search

- Betrachte alle Seiten, die für Anfrage *q* relevant sind oder die mit einer relevanten Seite verlinkt sind (In- und Outlinks).

  $\Rightarrow$ *$G_q(V_q, E_q)$ für Anfrage q*

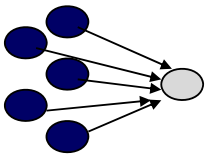
- *Zwei Typen von Webpages*:

  Hubs: Verlinken auf viele relevante Webpages (Authorities)

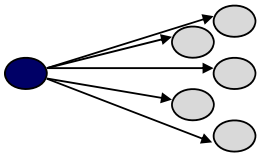
  Authorities: Werden von vielen guten Hubs verlinkt.

  => Jede Webpage ist zu gewissen Grad sowohl Hub als auch Authority.

  Für Webpage u sei h[u] der Hub-Score und a[u] der Authority-Score.



Eine gute Authority wird von vielen guten Hubs referenziert.



Ein guter Hub referenziert viele gute Authorities.

Berechnung von HITS:

- $\vec{a}$ Vektor aus Authority-Scores aller v $\in V_q$
- $\vec{h}$ Vektor aus Hub-Scores aller v $\in V_q$

- Berechnung über wechselseitige Iterationen : $\vec{a} = E^T \vec{h}$ (Authorityscore)

  $\vec{h} = E\vec{a}$ (Hubscore)

Vorgehen:

1. Bestimme alle relevanten Pages (Rootset).
2. Bestimme alle Pages, die Rootset verlinken.(Extendedset)
3. Iteriere über Hub- und Authority-Scores (Extended Set)
4. Sortiere Ergebnisse nach Hub-Score, Authority-Score oder Kombination.

Link Prediction

Gegeben: Ein Graph G(V,E) und 2 Knoten v,u $\in$ V mit (v,u) $\notin$ E.

Gesucht: Berechne ob der Link (v,u) in Zukunft entstehen wird oder ob er existiert und nur noch nicht beobachtet wurde.

Beispiele:

- Freundschaftsanfragen in sozialen Netzwerken
- unbekannte Proteininteraktionen
- Vorhersage von Kundenbewertung oder Käufen (Collaborative Filtering)

Feature-Basierte Link-Prediction

Grundidee: Paare von verlinkten Knoten haben meistens Eigenschaften, die für diesen Link verantwortlich sind.

Beispiele:

- Freunde haben häufig ähnliche Interessen.
- Co-Autoren forschen meist in ähnlichen Gebieten.
- Proteine haben komplementäre reaktive Zentren

$\Rightarrow$ Links enstehen nicht zufällig sondern durch die Kontakt von Objekten die komplementäre Eigenschaften haben.

$\Rightarrow$ Link Prediction lernet eine Funktion, die aus Paaren von Objektbeschreibungen die Existenz oder die Stärke eines Links vorhersagen

Formal: Sei u,v V und F(v),F(u) ihre Featurebeschreibung, dann versucht Link-Prediction, die Funktion: (F(v),F(u))-> L zu lernen.

(L ist dabei entweder {link, no link} oder [0,..max_Strength])

Strukturbasierte Vorhersage

Problem: Bei der Feature-Basierten Vorhersage wird nicht berücksichtigt, ob die Knoten Kontakt haben.

Beispiel:

- Personen mit gleichen Interessen lernen sich nicht kenn und verlinken sich daher nicht.
- Proteine können docken, kommen aber in der Natur nicht in den gleichen Zellen vor.

Lösungsansatz: Berücksichtige die Umgebung der Knoten v und u in G.

⇒ Haben beide Knoten ähnliche Nachbarn wächst die Wahrscheinlichkeit eines Kontakts und damit einer Verlinkung.

⇒ Beschreibe einen Knoten durch seine Adjanzenzliste im Graphen.

⇒ Wahrscheinlichkeit/Stärke eines Links kann durch die Ähnlichkeit der benachbarten Knoten vorhergesagt werden.

Link Prediction und Matrix-Faktorisierung

Gegeben: Graph G(V,E) mit Adjazenzmatrix A, des Graphen G(V,E) mit der Annahme das $a_{i,j} \in E$ unbekannt sind.

Annahmen:

- die Stärke der *Links* $U \subseteq A$ *ist* unbekannt
- Es gibt einen latenten *k*-dimensionalen Vektor-Raum mit $k < |V|$, der die Eigenschaften der Knoten beschreibt. => f(v) ist ein k-dimensionaler Vektor
- Die Stärke eines Links zwischen v und u besteht aus dem linearen Kernel K(u,v)

Vorgehen:

- Suche eine Zerlegung von in eine n×k Matrix B: $A' = BB^T$

  so dass $L(B) = \sum_{a_{i,j} \in A \setminus U} \left| a_{i,j} - a'_{i,j} \right|^2 = \sum_{a_{i,j} \in A \setminus U} \left| a_{i,j} - \left\langle b_{i,*}, b_{*,j} \right\rangle \right|^2$ minimiert wird.

Lösung: Gradientenverfahren über der Ableitung von L(B).

Anmerkung: Funktioniert auch für Two-Node Graphen (Kunde/Produkt)

Dense Subgraph Discovery

- Finde „dichte“ Subgraphen in einem Netzwerk.
- Definitionen für „dicht“:
- Cliquen (Vollständige Teilgraphen)
- Quasi-Cliquen (mindstens x % der möglichen Kanten sind vorhanden)
- relative Dichte zur Umgebung: Jeder Knoten in Teilgraphen G‘ hat mehr Links zu Knoten in G‘ als zu Knoten in G \ G‘.
- …
- Problem: Fast alle Formulierungen führen zu NP-harten Suchproblemen => heuristische Lösungen sind weit verbreitet.

Graph Clustering

- Klasse von Clustering Algorithmen die Datamenge als Graph betrachtet:
- Objekte = Knoten; Links sind mit Distanzen gelabelt.
- Häufig werde nicht alle Links verwendet: beschränken der Links auf die k nächsten Nachbarn oder ε-Umgebung
  => gerichtete und ungerichtete Graphen möglich
- Clustering über weighted k-Mincut: Zerlege Graphen in k möglichst gleich große, disjunkte Teilgraphen, wobei die Menge der durchtrennten Kanten minimert und die Kosten der getrennten Kanten maximiert werden sollen.
- Problem ist ebenfalls NP-hart.

Graph Clustering: Spektral Clustering

- bilde symmetrische Ähnlichkeitsmatrix : $S_{i,j} = sim(x_i, x_j)$
- Transformiere S in eine Graph Laplacian Matrix L:

$$L = I - D^{-\frac{1}{2}} S D^{-\frac{1}{2}} \qquad D_{i,j} = \begin{cases} \sum_k sim(x_i, x_k) & if \quad i = j \\ 0 & else \end{cases}$$

- nach Eigenwertzerlegung von L:
  - Eigenvektoren mit Eigenwert 0 repräsentieren Zusammenhangskomponenten des Graphen
  - Eigenvektoren enthalten lineare Gewichte für das Bilden eines Clusterrepräsentanten.

$$r_k = \sum_{i=1}^{|DB|} EV_i \cdot o_i$$

Fazit Graph Mining

- Graph-Mining bezeichnet viele neue Data Mining Tasks
  - Ranking wichtiger Knoten
  - Link-Prediction
  - Dichte-Teilgraphen und Community Detection
  - Frequent Subgraph Mining
- Clustering kann auch als Graphproblem formuliert werden
  - Dichtebasiertes Clustering: Finden von Zusammenhangskomponenten im Graphen wobei Links einem Ähnlichkeitsprädikat entsprechen.
  - Spectral Clustering
  - weighted k-Mincut Problem: Aufteilen einen Graphen in k Teilgraphen, so dass alle Teilgraphen möglichst gleich groß sind und die Summe der Gewichte aller „durchtrennten“ Kanten Minimal wird

Literatur

- Borgwardt K., Kriegel H.-P.: „Shortest-path kernels on graphs“. In Proc. Intl. Conf. Data Mining (ICDM 2005), 2005
- Borgwardt K.: „Graph Kernels“, Dissertation im Fach Informatik, Ludwig-Maximilians-Universität München, 2007
- Bunke, H. : „Recent developments in graph matching“. In ICPR, pages 2117–2124. 2000
- Gärtner, T., Flach, P., and Wrobel: „On graph kernels: Hardness results and efficient alternatives.“ Proc. Annual Conf. Computational Learning Theory, pages 129–143, 2003
- Wiener, H.: „Structural determination of paraffin boiling points“. J. Am. Chem. Soc., 69(1):17–20, 1947

Literatur

- Yan X., Han J.:“gSpan: Graph-based substructure pattern mining“, In ICDM, 2002.
- Kuramochi M., Karypis G.:“Frequent Subgraph Discovery“, In ICDM, 2001
- Brin S., Page L.:“The anatomy of a large-scale hypertextual Web search engine“, Computer Networks and ISDN Systems, Vol 30, Nr.1-7, S.107-117,1998
- Kleinberg J. M.:“Authoritative sources in a hyperlinked environment“, Journal of the ACM,Vol.46, Nr. 5, S. 604-632,1999
- Brandes U.:“A faster Algorithm for Betweenness Centrality“, Journal of Mathematical Sociology, 25(2):163-177, 2001