Graph Mining ( Graph basic )

왜 그래프 마이닝? = 99.9% 대부분의 데이터 셋은 속성간 그래프의 연결로 표현이 가능하다 이로 인하여 서로 다른 분야에서의 관계를 파악하기 쉽고 복잡한 것을 추상화 하기 쉽다

용어

데이터 엔티티 = 요소 ( 그래프에서의 점 )

데이터 관련 = 관련 ( 그래프에서의 선 )

그래프 엔티티 = G = ( V, E ,W )

* V = 노드 ( 데이터 )
* E = 엣지 ( 선의 개수 )
* W = 가중치 ( 선의 가중치 )

그래프의 정의

G = (V,E,W) / v = set of entity, e = set of relation, w = set of weight

그래프의 타입

|  |  |  |
| --- | --- | --- |
|  | Directed <a,b> | Undirected (a,b) |
| Weighted | 방향성 있지만 교환법칙 X | 방향성 없고 교환법칙 X |
| Unweighted  모든 가중치가 같은 경우 | 방향성 있고 교환법칙 O | 방향성 없지만 교환법칙 X |

Degree (N) = N노드에 연결되어있는 relation의 수

In-degree(N), out-degree(N) = N노드에서 나가는방향/들어오는방향의 relation 수

Dense = 그래프의 노드와 엣지가 몰려있다. 즉 그래프가 조밀조밀하게 모여있다

그래프의 표현

인접 행렬의 표현 (adjacent matrix) = M[i][j] = 1 -> i노드에서 j 노드 방향으로 가중치 1

트리 표현 = 모든 노드가 연결 되어 있어야 하고 cycle이 없을 경우에만 트리 표현 가능

* Isolated nodes = 연결이 안된 노드들
* Cycle = 다시 자기 자신으로 돌아오는 경로가 없어야한다.

그래프 분석 ( 모양 ).

* Degree distribution = 차수 분석 // P(k) =
  + P(k) = 차수가 k인 노드의 개수의 비율 = Nk(차수가 k인 노드 개수) / N
  + 주로 x축으로 노드의 개수, y축으로 P(k) 값으로 표현하여 그래프 그림
* Path Length = 길이 분석 // h
  + Path = 노드의 순차 -> Pn = {i0, i1, i2, …. In} = {(io,i1),(i1,i2)…. ,(in-1,in)}
  + Path는 같은 엣지를 여러 번 반복 지나갈 수 있다
  + Distance = Shortest Path length = A노드에서 B노드까지의 Path중에서 edge수가 가장 작은 것
    - Distance는 방향성 있는 그래프에서는 교환법칙 성립 X
  + Diameter = 그래프 내에서 Distance가 가장 큰 노드 쌍의 distance
  + Average path length = 전체 노드 쌍의 distance 평균
    - 방향성 X = 모든 노드쌍의 distance합 / 2\*nC2
    - 방향성 O = 모든 노드쌍의 distance합 / nC2
  + x축은 distance, y축은 path의 개수로 표현
* Clustering coefficient = 연결성 분석 // C
  + 자신의 친구들이 서로 친구인가?? 분석하는 것 / 즉 밀집도 분석
  + Ci = 노드 I의 clustering coefficient
    - Ci = 이웃노드끼리의 엣지 수 / 이웃 노드쌍 개수 = 2ei/ki(ki-1)
      * Ei = 이웃 노드끼리의 엣지 수
      * Ki = i노드의 차수
      * [0,1]사이의 값을 가진다.
  + C = Average clustering coefficient
    - C = Ci의 합 / N ( 노드 수 )
    - 높을수록 dense
  + x축은 차수, y축은 clustering coefficient 값으로 그래프 표현
* Connected components = 연결 컴포넌트 개수 분석 // s
  + Component = 노드의 그룹
  + Largest Connected component = 연결되어 있는 노드의 그룹중 가장 큰거
  + Connected component 찾는 방법
    - BFS = Breadth First Search 사용
  + x축은 컴포넌트에 포함된 노드의 개수, y축은 컴포넌트의 개수로 그래프 표현
  + GCC = Giant Connected Component = 가장 큰 연결 컴포넌트

그래프의 시각화 방법

* Log-Log Scale
  + 보기 힘든 그래프의 x축과 y축 모두 log스케일로 변환하는 방법
  + Heavy Tail
    - 전체 경향을 안따르는 노드들
    - 보통은 이것들을 다시 분석해야함

그래프 분석

* 위의 그래프 모양 4가지로 그래프 분석하였으면
* 이제 같은 노드, 같은 엣지의 수를 가지는 랜덤 그래프를 그려서 비교하면서 분석해

Graph Mining ( Random Graph )

랜덤 그래프 만드는 방법

* Erdos-Renyi ( 정규분포를 따르는 랜덤 그래프 ) = Gnp
  + Gnm = n개의 노드가 모두 연결되어 있다고 가정 한 후 m개의 엣지를 선택
  + Gnp = n개의 노드가 존재하고 모든 가능성의 엣지에 엣지 선택 확률 p 존재
    - Degree distribution = 정규분포를 따른다. = K/n
      * K = degree 평균
    - Path length = O(logN)
    - Clustering coefficient = K/n-1 ~~ K/n
      * Ci = 2ei/ki(ki-1), 이때 ei = p((ki(ki-1)/2) 왜냐면 같은 확률이니깐

그러므로 Ci = p = 차수의 평균 / 노드의 개수-1 = k/n-1

* + - Connected component = k가 1이상일 경우 GCC가 존재한다.
  + 중요한 부분은 Gnp에서는 Degree distribution과 Clustering coefficient에서 실 세계에서의 그래프와 크게 차이가 난다.

왜냐면 실제 네트워크에서는 몇몇개의 subgraph가 dense하게 묶이는 경향이 있지만 Gnp에서는 오직 같은 확률로만 엣지를 생성하기 때문에

* Small World
  + N개의 노드를 모두 연결시킨다 ( Tradic closure ) 그 후 하나의 엣지를 선택하여 연결 노드를 바꾸는 작업을 수행한다.
  + Rewire 작업
    - 엣지를 선택하여 연결을 바꿔주는 작업
    - 하면 할수록 clustering은 높아지고, diameter은 작아진다.
    - 실제 네트워크에서는 diameter은 낮고 clustering은 높고, rewire은 하면 초기에는 diameter이 급격히 낮아지다 멈추고 clustering이 어느정도 유지되다 급격히 낮아지므로 rewire을 어느정도 까지 계속 반복한다.

그래프의 자세한 분석에 사용되는 것들

* SubGraph
  + Direct, undirect에 따라서 subgraph의 개수가 달라진다.
  + 노드가 3개일떄 undirect subgraph는 2가지의 subgraph가 존재
  + Direct는 13가지의 subgraph가 존재
    - 이때 노드의 순서는 상관없고 모양만 중요함!!
  + 이렇게 subgraph를 통해서 어떤 subgraph들이 많이 나오는지에 따라 그래프의 전체적인 특징을 확인 할 수 있다.
    - 랜덤보다 많이 나오면 양수, 적게 나오면 음수로 표현
* Motif ( 그래프의 경향에 초첨을 둔 subgraph 분석 방법 )
  + 노드와 엣지의 수 뿐만 아니라 차수도 같게 해서 랜덤 그래프를 그려야 한다.
    - Random pairing방법 = 노드와 차수의 쌍을 (spoke)통해서 random pair을 수행한다.
    - Rewired 방법 = 한쌍의 엣지를 선택하여 출발자들끼리 또는 도달자들끼리 교차시켜주는 과정 이 과정을 반복해서 랜덤 그래프 생성
  + Pattern = 해당 그래프에서 발생한 subgraph
  + Recurring = 해당 패턴이 몇번 나오나?
    - 여기서 Overlapping을 허용해서 개수를 파악
  + Significant = 랜덤그래프보다 해당 패턴이 많이 나왔나 적게 나왔나?
    - Z-score의 값
      * Zi = i번째 패턴의 Z스코어
      * Zi = 실제 패턴i의 개수 – 랜텀 패턴i의 개수 / 랜텀 패턴i의 표준편차
    - Significan = SPi = Zi / 루트(모든 Zi^2의 합)
      * 여기서 분모 부분을 통해 -1~1값을 가지게 된다.
* Graphlet analysis ( 그래프 노드의 역할에 중점을 둔 subgraph 분석 방법 )
  + Graphlet = 단순 undirected 그래프들의 subgraph ( orbit )에서 노드의 역할별로 번호까지 붙여둔것들
  + GDV = graphlet degree vector
    - 노드별로 하나씩 가지는 벡터값
    - 그 노드가 포함되어있는 모든 subgraph에서 수행하는 역할
      * Ex) GDV(i) = (2,0,1,0) // 2,3노드까지의 graphlet까지만 분석하였을 때 I 노드는 0번노드역할 2번, 2번노드 역할 1번 수행

Motif와 Graphlet 찾는 방법

* Enumerating = ESU-Tree
  + Vsubgraph, Vextension가 존재
    - Vsub = 지금까지 구성한 sub그래프에 포함된 노드 집합
    - Vextension = Vsub에 추가되어 새로운 서브그래프를 이룰 가능성이 있는 노드 집합
    - Vextension에 포함되는 조건
      * 1. Vsubgraph의 모든 노드의 id보다 큰 id를 가진 노드
      * 2. Vsubgraph에 존재하지 않는 노드
  + ESU-tree = 그래프 찾는 과정에 구성이 되는 Tree
* Counting
  + Non-polynomial문제라서 추산으로 풀어야해
  + ESU-Tree에서 family끼리는 같은 사상으로 매칭될 확률이 높으니 family별로 추산하는 방향

Graph Mining ( Role&Centrality )

지금까지 그래프의 모양, 서브그래프들을 통해 그래프의 특징을 구했다면

이제는 중요한 노드를 추출, 노드에대해 어떤 특징을 가지는 노드들이 있는지 파악함

Role

* Role = 네트워트 상에서 노드들의 기능, 특징
  + Hub = 여러 노드들을 이어주는 노드
  + Members of cliques = dense한 노드들의 집합
  + Peripheral nodes (outlier) = 왕따
  + Role과 Group의 차이
    - Role에 포함된 노드 끼리는 그래프상에서 직접적인 연결이 없어도 괜찮다.
    - Group에 포함된 노드들은 그래프상에서 직접적인 연결이 존재한다.
  + Structual equivalence (구조적 동형)
    - In-degree, out-degree가 모두 같으면 구조적 동형이다.
    - 계산 방법은 인접행렬 상에서 row와 column이 모두 같으면 구조적 동형이다.
* Roix
  + 그래프 내에서 Role을 자동으로 생성해주는 툴
  + 엣지의 개수에 따라 선형적으로 계산횟수가 늘어나는 것이 존나 좋음
  + 진행 과정
    - Input = 인접행렬
    - Recursive Feature Extraction
      * Input = 인접행렬
      * Output = Node X Feature 행렬
      * 구해진 feature을 통해 새로운 feature을 만들어 내는 과정
      * Neighborhood
        + Local = 자기의 차수를 인풋으로 여러 수학계산으로 여러 feature생성
        + Egonet = 자기와 연결되어있는 노드의 차수들을 인풋으로 여러 수학 계산으로 feature 생성
      * Recursive
        + 지금까지 만들어진 feature들간에 여러 pair들을 통해 sum, mean 계산으로 feature 생성
        + 이때 반복하다보면 feature이 엄청 많아지는데 열별로 비교하여 같은 값을 가지는 열이 있으면 feature 삭제하면서 feature 단순화 (pruning 방법)
    - Role Extraction
      * Input = Node X Feature
      * Output = Node X Role , Role X Feature 즉 행력 2개
      * 여러 복잡한 알고리즘으로 결과물을 나오게한다.
* Centrality
  + Structual Role 분석중 한가지
  + 중요한 노드를 뽑아내는 방법
    - Degree centrality = 차수가 높은게 중요하다
      * Cd(i) = i노드의 차수 / 노드 개수 - 1
    - Closeness centrality = 이웃 노드와 많이 가까운 것들이 중요하다
      * Cc(i) = 노드 개수 -1 / 다른 노드들과의 거리의 합 = 모든 다른 노드와 평균 거리의 역수
    - Betweenness centrality = 더 많은 서브 네트워크와 연결되어있는 노드가 중요
      * Cb(i) = 다른 페어노드들의 path가 자기 자신지나는 count / 페어노드 경우의 수
    - Eigenvector centrality = 더 큰 네트워크들을 연결해주는 노드가 중요하다
      * 행렬의 고유값을 통해 계산하는거야 계산 방법은 복잡해서 설명 X
      * 연결이 많을수록, 연결된 노드의 연결이 많을수록 중요도 증가
  + Prestige
    - 방향성 있는 그래프에서의 Centrality

Graph Mining ( Communities )

* 커뮤니티?
  + 엣지들이 몰린 부분!
  + 엣지의 구분
    - Weak tie = 밀집되지 않은 지역에 존재한 edge
    - Strong tie = 밀집 지역의 edge
    - Overlap 계수가 높으면 strong, 낮으면 weak
      * 구하는 방법 = 엣지의 노드 기준 이웃노드 교집합 / 이웃노드 합집합
      * 계수가 0 = 로컬 브릿지가 1개만 존재한다.
      * 의미 = 해당 엣지를 안거치고 그 노드로 갈수 있는 비율
      * 낮은거부터 Overlap 계수를 자르면 두 노드간의 유일한 path가 지워질 확률이 크기 때문에 컴포넌트의 개수는 급격히 줄어들어
* Modularity
  + 군집성 = 해당 노드 집합의 밀집성
  + Q = 그룹별로 계산후 전체 다 더하기 (그룹내 엣지의 수 – 같은 노드갯수의 그룹내 기대 엣지수)
  + [-1,1]사이의 값을 가지게 되고 +이면 dense, - 이면 좆같은거
* 커뮤니티 찾는 방법
  + Newman 방법
    - Edge-Betweeness 이용하여 구한다.
    - 한계 = weighted, directed 그래프에서는 불가능하다. 느리다
    - 장점 = 쉽게 구현이 가능하고
    - Edge-betweeness
      * 계산 방법 (BFS 사용)
        + 1. 랜덤으로 노드 선택
        + 2. 그 노드기준 bfs 탐색으로 트리 구성
        + 3. Shortest path의 개수를 노드별 더함
        + 4. 맨 아래서부터 Node flow 계산 후 edge별로 EB 부여

Node flow = 1+자식 엣지들의 EB값

엣지별로 EB부여 비율 = 연결된 노드의 #shortest path 비율

* + - * + 5. 모든 노드에 대해 진행해서 EB값 다 더하기
    - 진행 과정
      * 1. edge-betweeness 계산
      * 2. 가장 큰값을 가지는 edge 제거
      * 3. 모든 엣지가 제거 될 때 까지 반복
      * 4. 제거의 역순으로 그룹화 하면서 modularity 계산
      * 5. Modularity가 최대가 될때까지 반복후 결과 리턴
    - Modularity가 제거되는edge의 수가 증가함에 따라 크게 증가하다 감소하는 이유
      * Modularity는 dense할수록 +값을 가지는데 edge-betweeness 가 높은거부터 제거하면 즉 weak tie를 제거하게 되므로 군집성이 증가하게 된다 하지만 제거하다 보면 strong tie를 제거하게 되고 이로 인해 군집성이 떨어지게 된다.
* Louvain 알고리즘
  + 탐욕 알고리즘을 사용
  + 장점 = 빠르다
  + 단점 = node에 대해 계산하는 순서에따라서 결과가 달라진다.
  + 진행 과정
    - Phase1
      * 색별로 분리하는 과정
      * 수행방법
        + 각 node를 커뮤니티에 할당
        + 노드별로 2가지 계산 수행

Node I 를 이웃 노드 j의 커뮤니티에 넣을때와 안넣을때의 modularity 차이 계산

가장 큰 차이를 보여주는 j쪽으로 i를 넣어줌

* + - * + 더 이상 modularity가 증가하지 않을때까지 수행
    - Phase2
      * 색별로 분리된거를 한 개의 super node로 줄여서 간단하게 표현하는 과정
    - Phase1, Phase2 뭉쳐서 PASS라 생각하면 오직 노드 1개만 남을때까지 수행해서 각 PASS의 output들의 modularity계산하여 가장 최적화된 결과를 리턴

Graph Visualization

그래프의 결과를 시각화 하는 여러 방법들에 대한 장단점 소개 및 중요하게 고려해야할 사항

* 그래프 시각화의 기본 원리
  + 자신의 연구에 진실성을 부여
  + 최소한의 자원으로 많은 정보를 보여주기
  + 보여주는 과정을 단계별로 구성하기
  + 시각화 그래프에 따라서 misleading이 가능하니 plot 선택도 중요함
* 그래프의 구분 방법
  + Discrete dimensions = 비교 카테고리 수
  + Continuous dimensions = 각 카테고리의 값들의 연속성
  + CD에 따른 구분
    - 1. 타겟값이 적을 경우
      * Column 히스토그램, pie chart, stacked column 히스토그램, 다이어그램
        + pie 차트는 비율을 강조할 때 사용
        + stacked column은 여러 legends에 대해 비율이 필요할 때
        + 다이어그램은 관계성을 나타낼 때
    - 2. 타겟값이 어느정도 있을 경우
      * Scatter Plots, Line Chart, Column Chart
        + Scatter Plots는 데이터가 많이지면 자원낭비 심함
        + Line Chart는 스케터에서 연결성까지 보여줄수 있음
        + Column Char (Bar Plot) 이거는 절대 분포에 대해 보여주지 않음
        + Strip chart, Box 플랏이 더욱 적절함 분포에 대해 보이고싶으면
    - 3. 타겟값이 여러 개일경우
      * Heatmap, bubble chart
        + 히트맵은 결과를 세세하게 나눌수 있고 grayscale로 주로 사용
        + 스케터에서 하나의 디멘젼을 추가한것으로 버블 size는 정확한 값이 아닌 비례값이다.
* 결론
  + 보여주려는 결과에 따라 차트를 잘 선택
  + 보여주고싶은 결과에 대해 최소한의 자원으로 보여줌
  + 단계별로 구상
  + Be true to your research