문서 기반 knowledge graph의 연결선 분포와 문서 주제 분석

권수훈

June 2022

Contents

1 abstract **2**

2 introduction **2**

3 Knowledge Graph Analysis **3**
3.1 data summary 3
3.2 Degree Distribution and heterogeneity 5
3.3 analysis results 6

4 Topic extraction **7**
4.1 data pre-processing 8
4.2 TextRank with Degree Distribution(k) 8

5 Results **11**

1 abstract

최근 Personal Knowledge Graph를 생성하고 관리할 수 있는 제품이 등장하고 있으나 적절한 Personalization 기능은 부족한 편이다. 따라서 본 연구는 Personalization을 위한 기반 작업으로써 Degree Distribution을 분석하여 문서 기반 개인 지식 그래프가 가진 특성을 파악해보고, 해당 특성에 기반하여 문서의 주제 분석을 다뤄보고자 한다.

2 introduction

조직 혹은 개인이 효율적으로 문서를 작성, 정리하고 이를 통해 빠르게 정보를 탐색하기 위해서 각종 생산성 도구들이 등장하고 있다. 대표적인 예로 Mendeley나 EndNote와 같은 서지 관리 프로그램과 Roam research를 필두로 한 Obsidian, Typed와 같은 document managements SaaS를 들 수 있다. 그리고 이러한 productivity tools들이 중점적으로 강조하는 사실은 작성하고자 하는 문서와 그 문서에 참고된 리소스들은 Graph 형태라는 것이다.

Roam research의 백서에 근거하여 설명하자면, 현재의 폴더 기반 구조가 캐비닛과 비슷하다면, graph 기반 문서 관리 툴은 네트워크 혹은 뉴런과 유사한 구조이다. 각 메모나 파일은 빈 폴더 내 존재하는 것이 아니라 상호 연결된 네트워크 내의 노드가 된다. 그리고 해당 노드가 다른 노드 시퀀스와 연결됨으로써 다른 노드와 대화하여 관계의 특성에 대한 정보를 주고받을 수 있다. 또한 네트워크는 동적이기 때문에 본래 의미를 변경하지 않고 컨텍스트를 유지할 수 있다. [1] 요약하자면, 문서들은 서로 연결됨으로써 그 특성과 맥락을 유지한채로 존재한다는 것이다.

실제로 위 사고를 네트워크로 구성해보면 문서와 해당 문서를 작성, 이해하는데 필요한 맥락 정보인 참고자료(Resource)를 노드로 가지는 네트워크가 구성되며, 다음과 같은 규칙이 성립한다.

1. 하나의 문서는 웹 페이지, pdf, text 파일 등의 Resource를 통해 작성된다.
2. 완성된 문서는 또 다른 문서의 Resource로 활용될 수 있다. (Document as a Resource)

[1] If current tools resemble filing cabinets, Roam is more akin to the nodal networks in telecommunications, or the neurons in the human brain. Rather than existing in a vacuum, each note or file becomes a node in an interconnected graph of ideas. A single node may simultaneously hold positions in several different sequences, hierarchies or file paths, and can 'talk' to other nodes, communicating information back and forth about the nature of each relationship. The network is dynamic, so updates and revisions are populated across the entire graph simultaneously. Individual nodes or branches within the network can be forked as required, allowing a new pathway to deviate without changing the original meaning. "Road Research White pager", https://roamresearch.com/#/app/help/page/dZ72V0Ig6

이를 도식화하자면 Figure 1과 같다.

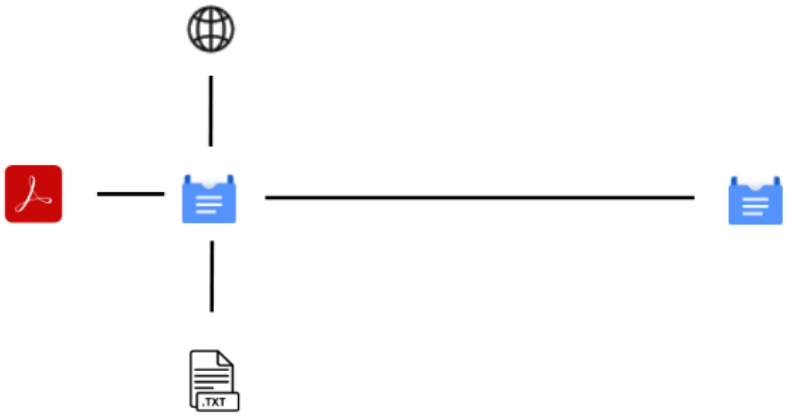

Figure 1: 문서 네트워크의 최소 단위

이러한 그래프 하에서는 많은 기능이 개발될 수 있다. 단순히 네트워크를 조감하는 것만으로도 개별 문서에서 보이지 않는 패턴 혹은 클러스터를 식별하는데 도움이 될 수 있으며 이는 종종 Serendipity를 유도하기도 한다. 다른 가능성으로는 이 구조를 지식 그래프의 데이터 모델 중 하나인 RDF triple의 관점에서 본다면 Link Prediction이 가능하다는 것이다. Document(subject)가 Resource(object)와 Link(predicate)하고 있다고 정리할 수 있다. 그렇다면 우리는 subject와 predicate를 기반으로 object를 자동으로 도출할 수 있을 것이다. 이는 곧 특정 문서 혹은 리소스에 대한 추천으로 이어진다.

3 Knowledge Graph Analysis

이 섹션에서는 문서 기반 지식 그래프가 어떤 특성을 가지고 있는지를 분석해보고자 한다. 지식 관리 서비스를 운영하고 있는 국내 업체의 협조를 받아 DB snapshot을 연구 목적으로 제공 받아 진행되었다. 네트워크를 구성하는 노드는 문서와 문서를 작성하는데 참고한 리소스로 구분되며 리소스는 텍스트(text), 파일(file), 웹페이지(url), 문서(document as a resource)로 구성된다. 이해를 위하여 실제로 구성된 네트워크의 일부를 첨부하자면 Figure 2와 같다. 지속적인 서비스 제공이 아닌 연구 목적이므로 EtLT 파이프 라인 구축 등 데이터 엔지니어링에 관련된 작업은 생략하였다.

3.1 data summary

일반적으로 network는 방향성과 가중치의 유무에 따라 구분이 된다. 현실에서 가장 많이 보이는 종류의 그래프는 direct weighted network, 즉, 방향성과 가중치가 존

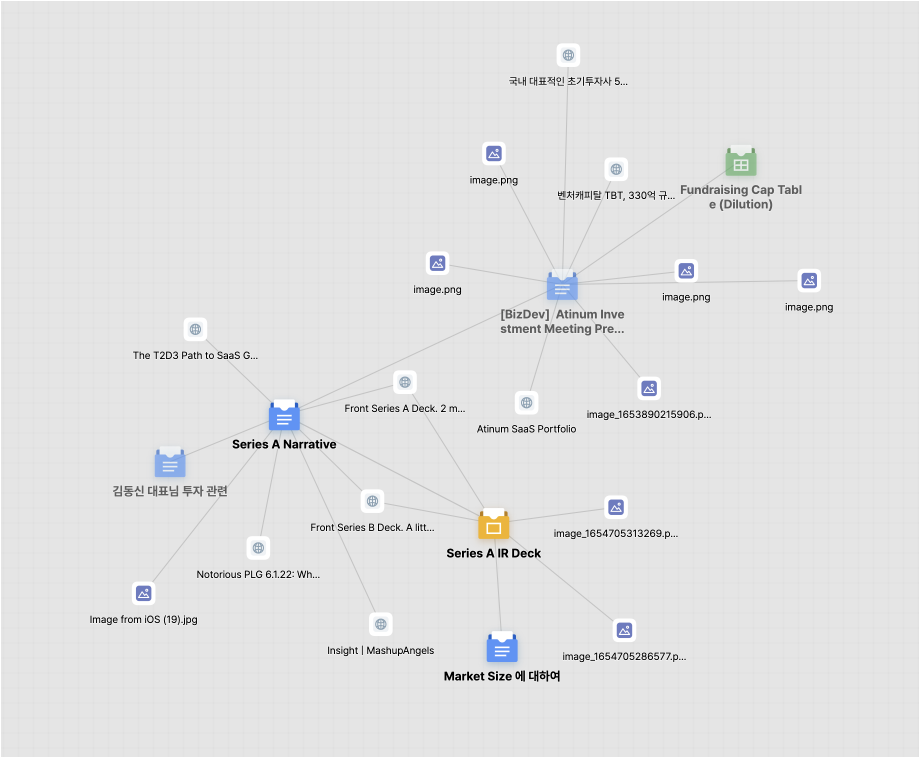

Figure 2: 문서 기반 지식 네트워크

재하는 그래프이다. 그러나 지식 네트워크는 가중치는 존재하지 않고 방향성은 그 종류에 따라 유무가 다르다. 본 연구에서 분석하고자 하는 문서 기반 지식 네트워크에서는 한 노드에서 다른 노드로 링크하게 되면 자동으로 backlink가 생기므로 양방향으로 소통하게 된다. 따라서 본 연구에서 분석하는 네트워크는 방향성과 가중치가 존재하지 않는 네트워크를 가정한다.

또한 문서 기반 지식 그래프의 특성상 리소스는 여러 문서에 링크될 수 있지만 리소스 자체가 다른 리소스와 연결될 수는 없다. 이러한 특성 때문에 자연스럽게 star형 네트워크 토폴로지를 가지게 된다.

분석의 대상이 되는 데이터에서 작성된 문서의 갯수는 31,694개이며 문서를 작성하는데 참고한 리소스는 문서 자신을 포함하여 158,240개이다. 이 갯수가 곧 노드의 갯수라 할 수 있다. 리소스의 타입별 갯수는 Table 1과 같다.

Type	Quantity
document	31694
url	84387
file	33286
text	8873

Table 1: 리소스의 갯수

Type	Quantity
count	182524
mean	0.522616
std	2.877597
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	335

Table 2: degree description

이러한 노드들 사이의 연결선인 엣지의 갯수는 114400개이다. 노드와 링크의 수를 파악했으므로 본 네트워크에서의 조밀도(link density)를 구할 수 있다.

$$d = \frac{L}{L_{max}} = \frac{2L}{N(N-1)}$$

해당 그래프가 완전 네트워크(complete network)일 경우의 링크의 갯수인 L_{max}를 분모로, 실제 링크의 갯수를 분모로 하여 조밀도를 산출한 결과로 대략 $6.8e-06$가 도출되었다. 이는 문서 기반 지식 네트워크가 매우 sparse한 네트워크란 것을 암시한다. 이는 대규모 네트워크에서 흔히 보이는 성긴 정도에 해당한다. 페이스북의 조밀도는 $d \approx 1e-7$, 인터넷 라우터의 조밀도는 $d \approx 3e-5$ 정도로 추산된다. [1]

3.2 Degree Distribution and heterogeneity

본 연구에서는 네트워크의 전체적인 특성에 관심이 있으므로 연결선 수(degree, k)의 분포를 중심으로 살펴보고자 한다.

각 노드에 연결된 엣지의 갯수를 집계한 후 간단한 통계치를 Table 2에 정리하였다. 75% 백분위 수에 이르기까지도 연결선 수는 0에 수렴한다. 일반적으로 자연계의 그래프의 연결정도 중심성(degree centrality)은 멱함수적 분포를 가지고 있다는 특성을 고려할 때, 문서 기반 지식 네트워크에서는 소수의 노드가 많은 링크를 가지는 hub node인 반면 대다수의 노드는 연결선 수가 없는 singleton node에 해당할 것이라고 추측할 수 있다.

degree	node count
0	168003
1	2570
2	1630
3	1354
4	1312
56	1
57	1
58	1
59	1
335	1

Table 3: degree value counts

또한, 특정 연결선 수를 가진 노드의 갯수로 집계해보았을 때 연결선 수가 0인 노드가 167996개인 반면 연결선 수가 1인 노드는 2570개로 급격히 줄어드므로 연결선 수의 분포가 불균일(heterogeneous)하며 편차가 심할 것이라고 추측할 수 있다.

불균일한 정도를 정량적으로 파악하기 위하여 불균일도heterogeneity κ를 확인하기 위해서는 다음과 같은 공식을 활용한 결과 31.31622이 산출되었다.

$$\kappa = \frac{\langle k^2 \rangle}{\langle k \rangle^2}$$

여기서 $\langle k \rangle$는 네트워크 전체의 평균 연결선 수(average degree in the network)를 의미한다. 평균 연결선 수 $\langle k \rangle = \frac{\sum_i k_i}{N}$ 임을 고려할 때, $\langle k^2 \rangle = \frac{k_1^2 + ... + k_N^2}{N}$ 이다.[2]

이러한 불균일도로 추정할 수 있는 것은, 만약 해당 네트워크가 정규 분포를 이루었다면 연결선 수의 제곱값은 k^2 부근에 존재할 것이므로 $\langle k^2 \rangle \approx \langle k \rangle^2$일 터이므로 κ는 1에 가까워야 한다. 그러나 약 31과 같이 큰 값이 도출되었다는 것은 특정 노드의의 k가 매우 높은 형태의 불균형한 분포를 이루고 있다는 것을 의미한다.

3.3 analysis results

앞서 3.1과 3.2에서 살펴본 데이터의 특성, 수치, 시각적인 그래프를 통해서 아래와 같은 사실을 정리할 수 있다.

1. 문서 기반 지식 네트워크의 특성상 네트워크 토폴로지가 대부분 star형이며 불균일적이다.

   (a) link density가 낮은 성긴 네트워크이고, 수치상으로도 singleton node가 높은 비중을 차지하고 있다.

   (b) 소수의 hub node들이 존재하며 이들이 형성하는 네트워크의 토폴로지 또한 star형으로 비슷하므로 단순히 그래프의 모양의 유사성을 기반만을 근거로 문서 주제의 유사성을 보장할 수 없다. 따라서 RecSys나 topic

extraction에 있어 graph2vec과 같이 노드와 엣지의 유사성을 기반으로 한 임베딩만으로는 정확하지 못한 결과를 낼 가능성이 높다.

(c) 문서에 첨부된 리소스는 hub node가 되지 못한다. 첨부된 리소스는 문서를 위한 일종의 주석과도 같은 역할이므로 해당 문서의 중요성을 판단하는 가중치로 활용할 수 있을 것이다.

4 Topic extraction

본 연구의 주제가 personalized를 위한 문서 기반 지식 그래프의 분석인 만큼 topic extraction을 진행해보고자 한다. topic이나 keyword는 문서 집합을 분류하는데 유용한 시작점이 될 수 있으며 domain-specific한 단어 사전을 구축하는데도 유용할 수 있어 지식 네트워크의 개인화의 기반이 될 수 있다.

본 연구에는 별도의 학습이 필요 없이 통계적으로 중요 단어를 추출할 수 있는 textrank 알고리즘을 적용하여 개별 문서의 핵심 단어를 추출하고 더 나아가 문서의 연결선 수를 가중치로 반영하여 문서 네트워크의 핵심 단어를 추출해보고자 한다.

2004년에 제시된 textrank 알고리즘은 The pagerank citation ranking: Bringing order to the web. (Page, Lawrence, et al., 1999)에 제시된 pagerank 알고리즘을 단일 문서에 적용하여 문서 내 중요한 단어 혹은 문장을 추출하는 알고리즘이다.

pagerank는 각 웹 페이지를 노드로, 웹 페이지 내 하이퍼링크를 엣지로 간주하며 이러한 그래프를 마르코프 체인으로 간주하여 수렴할 때 까지 연산을 반복하는 방식으로 웹 페이지의 중요도를 측정한다. [2] [3]

textrank가 제시된 textrank: Bringing order into text. (Mihalcea, Rada, and Paul Tarau., 2004)[4] 에서는 pagerank를 문서에 적용하기 위한 방법으로 단어를 노드로 보고 특정 window 내에 단어가 존재하는 co-occurrence를 기반으로 그래프를 구성한다. [4]

[2] 마르코프 체인의 수렴에 대한 수학적 설명은 [3]을 참고.

[3] 웹 페이지 u의 페이지 랭크 $R(u)$는 다음과 같이 계산될 수 있다.

$$R(u) = (1-d)\frac{1}{N} + d\sum_{v \in B_u} \frac{R(v)}{N_v}$$

d는 damping factor로 하이퍼링크를 따라 탐색을 지속할 확률이며 1-d는 jumping factor로 forward link가 존재하지 않거나 cyclic한 네트워크의 탈출을 위해 새로운 문서에서 탐색하는 행동을 모사하기 위해 도입된 항이다. 따라서 damping factor는 웹 페이저 u를 가리키는 문서 집합 B_u와 연관된다.

[4] 원문을 옮기자면 다음과 같다.

1. Identify text units that best define the task at hand, and add them as vertices in the graph.
2. Identify relations that connect such text units, and use these relations to draw edges between vertices in the graph. Edges can be directed or undirected, weighted or unweighted.
3. Iterate the graph-based ranking algorithm until convergence.
4. Sort vertices based on their final score. Use the values attached to each vertex for

4.1 data pre-processing

textrank를 적용하기 위하여 텍스트 데이터를 정제하고 토크나이징할 필요가 있다. 공백과 특수문자, 이모티콘, 영문을 제거함으로써 데이터를 정제하였고 반복되는 단어를 normalize한 후 sentence segmentation을 실시하였다. 그 이후 문장 별로 품사 태깅을 진행하여 명사와 형용사만을 추려내었다. 구현 논문에 따르면 명사과 형용사만을 대상으로 할 때 가장 퍼포먼스가 좋았다고 한다.[5]. 그 이후 불용어를 삭제하였다. [6]

4.2 TextRank with Degree Distribution(k)

문서에서 키워드를 추출한 후 연결선수 k를 기반으로한 중심도(degree centrality)를 가중치로 고려하고자 한다. 문서에서 도출된 태그에 degree centrality를 곱하여 값을 구할 수 있다.

그 첫 단계로 토큰화된 문장내 단어를 id와 매핑하여 index를 부여한다.

```
class WordIndexer():
    def __init__(self, tokenized_doc_series):
        self.tokenized_doc_series = tokenized_doc_series
        self.tokens = []
        self.word_count = defaultdict(int)
        self.idx_to_vocab = []
        self.vocab_to_idx = {}

    def get_nouns_tokens(self):
        for _, document in self.tokenized_doc_series.iteritems():
            self.tokens.append(document)

        return self

    def word_counting(self):
        for noun in [token for sent in self.tokens for token in
    sent]:
            if noun in self.word_count:
                self.word_count[noun] += 1
            else:
                self.word_count[noun] = 1
        return self

    def mapping_vocab_to_id(self, min_count = 2):
        threshold_word_count = {word: count for word, count in self
    .word_count.items() if count >= min_count}
        self.idx_to_vocab = [w for w, _ in sorted(
    threshold_word_count.items(), key=lambda x:-x[1])]
        self.vocab_to_idx = {vocab:idx for idx, vocab in enumerate(
    self.idx_to_vocab)}
```

ranking/selection decisions.

[5]"the best performance was achieved with the filter that selects nouns and adjectives only", Mihalcea, Rada, and Paul Tarau. "Textrank: Bringing order into text." Proceedings of the 2004 conference on empirical methods in natural language processing. 2004.

[6]이 과정의 구체적인 구현은 https://github.com/DarrenKwonDev/bachelor_thesis 참고

```
        return self.idx_to_vocab, self.vocab_to_idx
```

Listing 1: word id mapping

이제 window내 토큰이 존재하는 지에 대한 여부를 수치화히기 위하여 동시 발생 행렬(co-occurrence matrix)을 생성한다. 이러한 처리는 textrank 알고리즘과 별도로 단어의 의미가 주변의 단어에 의해 형성된다는 분포 가설(distributional hypothesis)에 의거한 것이다. 단, 윈도우의 크기는 원본 논문에 근거하여 2와 10 사이의 값으로 세팅한다.

"We are using a co-occurrence relation, controlled by the distance between word occurrences: two vertices are connected if their corresponding lexical units co-occur within a window of maximum N words, where N can be set anywhere from 2 to 10 words. [4]

```
class CooccurMatrix():
    def __init__(self, tokens, vocab_to_idx, window = 5):
        self.tokens = tokens
        self.vocab_to_idx = vocab_to_idx
        self.window = window
        self.cooccurrence_matrix = None

    def dict_to_mat(self, d, n_rows, n_cols):
        rows, cols, data = [], [], []
        for (i, j), v in d.items():
            rows.append(i)
            cols.append(j)
            data.append(v)
        return csr_matrix((data, (rows, cols)), shape=(n_rows,
    n_cols))

    def create_co_occurrence_matrix(self):
        counter = defaultdict(int)
        for _, tokens_i in enumerate(self.tokens):
            vocabs = [self.vocab_to_idx[w] for w in tokens_i if w
    in self.vocab_to_idx]
            n = len(vocabs)
            for i, v in enumerate(vocabs):
                if self.window <= 0:
                    begin_index, end_index = 0, n
                else:
                    begin_index = max(0, i - self.window)
                    end_index = min(i + self.window, n)
                for j in range(begin_index, end_index):
                    if i == j:
                        continue
                    counter[(v, vocabs[j])] += 1
                    counter[(vocabs[j], v)] += 1

        counter = {k:v for k,v in counter.items()}
        n_vocabs = len(self.vocab_to_idx)
        return self.dict_to_mat(counter, n_vocabs, n_vocabs)
```

Listing 2: co-occurrence matrix

단어의 지정 window내 존재하는 단어는 상호적으로 존재하므로 동시 발생 행렬은 대칭이 된다.

```
def is_matrix_symmetric(matrix):
    return (matrix==matrix.T).toarray().all()

is_matrix_symmetric(C) # True
```

Listing 3: co-occurrence matrix symmetric check

R이 수렴할 때까지 반복 수행을 해야 한다. 일반적으로 20~30번의 iteration으로 본다. "usually for 20-30 iterations, at a threshold of 0.0001" [4]

```
    def pagerank(x, damper_factor=0.85, iter_count=30):
        A = normalize(x, axis=0, norm='l1')
        R = np.ones(A.shape[0]).reshape(-1,1)
        bias = (1 - damper_factor) * np.ones(A.shape[0]).reshape
    (-1,1)

        for _ in range(iter_count):
            R = damper_factor * (A * R) + bias

        return R

    def get_keywords(C, k = 5):
        R = pagerank(C, 0.85, 30).reshape(-1)
        idxs = R.argsort()[-k:]
        keywords = [(idx_to_vocab[idx], R[idx]) for idx in reversed
    (idxs)]
        return keywords
```

Listing 4: co-occurrence matrix symmetric check

개인적으로 정리한 자연어 처리 관련 문서에 대하여 키워드 분석한 결과 중 상위 10개의 결과는 다음과 같았다.

[(('단어', 'Noun'), 15.799525519527279), (('추출', 'Noun'), 4.2487027933104144), (('있다', 'Adjective'), 3.4571654360080495), (('분석', 'Noun'), 2.9014140005551052), (('문서', 'Noun'), 2.880536781130387), (('기반', 'Noun'), 2.7888458051264213), (('한국어', 'Noun'), 2.654256351979765), (('사용', 'Noun'), 2.6013156458118605), (('표현', 'Noun'), 2.525392613862564), (('등장', 'Noun'), 2.336757034648675)]

이 과정은 모든 문서에 적용한 후 집계하여 시각화하면 Figure 3 같은 결과를 얻을 수 있다.

Figure 3: 가중치를 반영한 word cloud

5 Results

본 연구는 문서가 홀로 존재하는 것이 아니라 다양한 맥락에서 구성되고, 그 맥락 정보를 활용하자는 시도였다. textrank를 통한 문서의 주제를 대표하는 키워드 추출 외에도 그래프 기반 뉴럴 네트워크와의 결합 등 다양한 조합과 연구가 이루어질 수 있을 것이라 기대한다.

References

[1] F. M. et al., "A first course in network science," in *A First course in Network Science*, 2020, p. 47.

[2] ——, "A first course in network science," in *A First course in Network Science*, 2020, p. 112.

[3] A. Freedman, "Convergence theorem for finite markov chains," *Proc. REU*, 2017.

[4] R. Mihalcea and P. Tarau, "Textrank: Bringing order into text," in *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004, pp. 404–411.