Report

Class 데이터 과학

Title | PageRank

학 과 명	컴퓨터공학과
교 수 명	이영석 교수님
학 번	201601989
이 름	김 진 섭
제 출 일	2019. 06. 14

충남대학교
Chungnam National University

목 차

과제 목표

[과제 수행 목표]

1. 성능 비교

- PageRank를 수행할 때 Dictionary를 활용하는 방법과 Numpy를 활용하는 방법에 대한 성능 비교를 수행한다.
- 얼마나 성능 차이가 나는지, 왜 그렇게 성능 차이가 나는지 이해한다.
- 두 가지 방법의 계산 시간을 표시하고 비교 설명을 수행한다.

2. 영화 Top 10 구하기

- Dictionary와 Numpy를 활용하여 계산된 PageRank를 이용한다.(총 두 개)
- 가장 Rank가 높은 영화 10개를 보여준다.
- 영화의 번호, 영화 이름, Rank점수를 출력하여준다.

3. Top 10 시각화

- Dictionary와 Numpy를 활용하여 계산된 PageRank를 이용한다.(총 두 개)
- 가장 Rank가 높은 영화 10개의 노드를 시각화하여준다.
- pyvis와 networkx 라이브러리를 활용하여 시각화를 할 수 있다.
- 시각화한 결과를 .html로 출력한다.

과제 진행을 위한 초기 단계

[사용 데이터 : github에 제공]
- 영화 정보 데이터 - movies.csv
- 배우 데이터 - actors.csv
- 영화별 배우 정보 - casting.txt
- 사용자 평점 정보 - ratings.csv

[기본 데이터 로드]

In [1]:
```
import sys
import pickle
from pprint import pprint
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from data.make_casting_graph import oneway_to_bidirected_graph
from scipy.sparse import csc_matrix
import time
from pagerank import pagerank
from sklearn.preprocessing import normalize
from pyvis.network import Network
```

과제를 진행하기 위해 필요한 모듈 import

In [2]:
```
# create idx to num comments
with open('./data/ratings.csv', encoding='utf-8') as f:
    docs = [line.strip().split(',') for line in f.readlines()[1:]]
    _idx2numcomments = {movie_idx:int(num) for num, movie_idx in docs}
```

In [3]:
```
# pre defined casting weight graph
with open('./data/casting_graph.pkl', 'rb') as f:
    graph = pickle.load(f)
```

In [4]:
```
# create idx to actor name function
with open('./data/actors.csv', encoding='utf-8') as f:
    next(f)
    docs = [line.split(',') for line in f.readlines()[1:]]
    # English name if exist else Korean name
    _idx2actor = {doc[0]:doc[1] for doc in docs}
```

In [5]:
```
with open('./data/movies.csv', encoding='utf-8') as f:
    next(f)
    docs = [line.split(',') for line in f.readlines()[1:]]
    _idx2movie = {doc[0]:doc[1] for doc in docs if len(docs)}
```

In [6]:
```
idx2movie = lambda idx: _idx2movie.get(idx, 'Unknown')
idx2actor = lambda idx: _idx2actor.get(idx, 'Unknown')
idx2numcomments = lambda idx: _idx2numcomments.get(idx,0)
```

In [7]:
```
g = oneway_to_bidirected_graph(graph)
```

과제 진행을 위해 기본 데이터 로드
idx2movie : 영화 번호를 활용해 영화 제목을 가져올 수 있음
idx2actor : Actor 번호를 활용해 Actor의 이름을 가져올 수 있음
g : ActorNode와 MovieNode에 대한 그래프이며, 어떠한 영화에 어떠한 배우들이 연결되어있는지 확인이 가능
(PageRank관련 코드는 Github에 제공)

[과제1] 성능 비교

(Dict를 활용한 경우)

Dict를 이용한 경우

```
bias = {node:(idx2numcomments(node.split()[1]) if node[0] == 'm' else 0) for node in g}
_sum = sum(bias.values())
bias = {node:b / _sum for node, b in bias.items()}

Start = time.time()
rank_Dict = pagerank(g, bias = bias, df = 0.15, max_iter = 30, converge_error = 0.001, verbose = 1)
Finish = time.time()
print("Dict를 이용한 PageRank 걸린 시간 : ",round(Finish-Start,3))
```

```
Iteration = 1, diff = 0.6745935594038653, sum = 1.000000000000003
Iteration = 2, diff = 0.5133755765513065, sum = 1.000000000000062
Iteration = 3, diff = 0.4070843471025293, sum = 1.000000000000078
Iteration = 4, diff = 0.3288114569448776, sum = 1.000000000000004
Iteration = 5, diff = 0.2690000626169725, sum = 1.000000000000069
Iteration = 6, diff = 0.2217292304566567, sum = 0.999999999999992
Iteration = 7, diff = 0.1837276549693113, sum = 0.999999999999958
Iteration = 8, diff = 0.1529064807655553, sum = 1.000000000000047
Iteration = 9, diff = 0.1275639162436215, sum = 0.999999999999909
Iteration = 10, diff = 0.1067656357170639, sum = 0.999999999999944
Iteration = 11, diff = 0.0894733554563145, sum = 1.000000000000075
Iteration = 12, diff = 0.0751701431966287, sum = 1.00000000000001
Iteration = 13, diff = 0.0631852881144808, sum = 0.999999999999933
Iteration = 14, diff = 0.0532060909784067, sum = 0.999999999999913
Iteration = 15, diff = 0.0448304779270671, sum = 1.000000000000069
Iteration = 16, diff = 0.0378108514646868, sum = 1.000000000000069
Iteration = 17, diff = 0.0319123472901759, sum = 1.000000000000033
Iteration = 18, diff = 0.0269472572380865, sum = 1.000000000000056
Iteration = 19, diff = 0.0227701840628960, sum = 1.00000000000084
Iteration = 20, diff = 0.0192479674465808, sum = 1.000000000000036
Iteration = 21, diff = 0.0162770372749414, sum = 1.000000000000004
Iteration = 22, diff = 0.0137702888687806, sum = 0.999999999999948
Iteration = 23, diff = 0.0116527587512553, sum = 0.999999999999926
Iteration = 24, diff = 0.0098642075849924, sum = 1.000000000000062
Iteration = 25, diff = 0.0083518602489720, sum = 1.00000000000006
Iteration = 26, diff = 0.0070739230412209, sum = 0.999999999999979
Iteration = 27, diff = 0.0059922222715457384, sum = 0.999999999999997
Iteration = 28, diff = 0.0050772942421771106, sum = 1.000000000000002
Iteration = 29, diff = 0.0043028047100510720, sum = 1.000000000000004
Iteration = 30, diff = 0.0036469409233534757, sum = 0.999999999999993
Dict를 이용한 PageRank 걸린 시간 :  0.512
```

Dictionary를 활용해 PageRank를 한 결과이다.

time을 활용하여 걸린 시간을 측정하였다.

bias값은 리뷰가 많이 작성된 리뷰일수록 중요도가 커지게 하기 위한 값으로 PageRank에서 활용한다.

우선, PageRank를 계산하기 위하여 bias값을 Dictionary로 구해주고, 약 30번 정도 반복하여 돌면 error가 0.001에 가깝게 내려가기 때문에 iter값을 30으로 놓고 pagerank를 수행하여준다.

Dictionary를 활용하여 걸린 시간은 약 0.512초 이다.

(Numpy를 활용한 경우)

```
nodes = set(g.keys())
idx2node = list(sorted(nodes))
node2idx = {node:idx for idx, node in enumerate(idx2node)}

bias = np.asarray([b for node, b in sorted(bias.items(), key = lambda tp:node2idx[tp[0]])])
print(bias.shape)

rows =[]
cols =[]
data =[]

for from_node, to_dict in g.items():
    from_idx = node2idx[from_node]
    for to_node, weight in to_dict.items():
        to_idx = node2idx[to_node]
        rows.append(from_idx)
        cols.append(to_idx)
        data.append(weight)
A = csc_matrix((data, (rows, cols)))
print(A.shape)
```

```
(6154,)
(6154, 6154)
```

Dictionary로 생성된 그래프를 매트릭스로 만들어주는 과정의 코드이다.

```
max_iter = 30
df = 0.85

Start = time.time()
ir = 1/A.shape[0]
rank_Numpy = np.asarray([ir] * A.shape[0])

for n_iter in range(1, max_iter + 1):
    rank_new = A.dot(rank_Numpy)
    rank_new = normalize(rank_new.reshape(1,-1), norm = 'l1').reshape(-1)
    rank_new = df * rank_new + (1-df) * bias
    diff = abs(rank_Numpy-rank_new).sum()
    rank_Numpy = rank_new
    print('iter {} : diff = {}'.format(n_iter, diff))
Finish = time.time()
print("Numpy를 이용한 PageRank 걸린 시간 : ",round(Finish-Start,3))
```

```
iter 1 : diff = 0.1685245368865779
iter 2 : diff = 0.1235344416788289
iter 3 : diff = 0.11717242074154521
iter 4 : diff = 0.08676250638774644
iter 5 : diff = 0.08106650827175174
iter 6 : diff = 0.06044614044638538
iter 7 : diff = 0.05589527869039322
iter 8 : diff = 0.04188475454126574
iter 9 : diff = 0.038452782327255894
iter 10 : diff = 0.028909517190488 6
iter 11 : diff = 0.026405522194198443
iter 12 : diff = 0.019944863886447 59
iter 13 : diff = 0.018111372899163 91
iter 14 : diff = 0.013753287448751986
iter 15 : diff = 0.012408911428306675
iter 16 : diff = 0.009469243738374537
iter 17 : diff = 0.008494000468005527
iter 18 : diff = 0.006511648928942716
iter 19 : diff = 0.005809774127703195
iter 20 : diff = 0.004473307017566352
iter 21 : diff = 0.003971296705335752 5
iter 22 : diff = 0.003070457850610517 3
iter 23 : diff = 0.002715284598268786 6
iter 24 : diff = 0.002106149459828414
iter 25 : diff = 0.001857703937423409 1
iter 26 : diff = 0.001443802195180850 3
iter 27 : diff = 0.001270456142678351 4
iter 28 : diff = 0.000989257655965191 4
iter 29 : diff = 0.000868553676668131 7
iter 30 : diff = 0.000677506241060136
Numpy를 이용한 PageRank 걸린 시간 :  0.026
```

내적을 활용하여 PageRank를 계산한다. 30회 수행함을 알 수 있다.
Numpy를 활용한 결과 0.026초가 걸렸음을 알 수 있다.

(계산 속도 비교)
- Dictionary를 활용하여 걸린 시간 약 0.512초
- Numpy를 활용하여 걸린 시간 약 0.026초

Dictionary를 활용하는 것이 Numpy를 활용하는 것 보다 훨씬 오래 걸림을 알 수 있다.

그 이유는 Dictionary를 활용하면 한 영화와 그 영화의 액터들을 모두 탐색하며 들어가고, 모두 하나씩 계산하고 Sum을 해주는 과정이 수행되는데, 그 과정이 약 n 만큼이 수행된다.

하지만 Numpy를 활용하여 행렬로 처리하고 내적을 사용하면 한 번에 같은 행을 모두 곱할 수 있다.

따라서, Numpy를 활용하여 PageRank를 계산하는 방법이 훨씬 빠르다.

[과제2] 영화 Top 10

Numpy와 Dictionary를 활용해 계산한 Rank값이 높은 영화 Top10 뽑기

(Dictionary 활용)

Dict를 이용한 방식

```
num = 0
for movie in sorted(rank_Dict.items(), key = lambda x: x[1], reverse=True):
    if num == 10 :
        break
    if movie[0].split()[0] == 'movie':
        print(movie[0].split()[1],idx2movie(movie[0].split()[1]),movie[1])
        num += 1
    else:
        continue
```

```
161967 기생충 0.0032033878121671224
167651 극한직업 0.0014303471787626468
175322 마녀 0.0011565783119412997
156464 보헤미안 랩소디 0.0011527961465662747
130966 부산행 0.001098819013448319
177483 배심원들 0.0009469824923736168
174065 걸캅스 0.0009354687095915042
37886 클레멘타인 0.0009182492132450038
154449 리틀 포레스트 0.0009182174784566 3
163788 알라딘 0.0007997936563664337
```

Top10을 뽑는 방법이다. num값을 0부터 시작하여 10개를 뽑아낸다. 위에서 Dictionary로 PageRank를 돌린 값의 item들을 Sorting하여 수행한다.

그런데, 여기서 값을 찾을 때, Node는 MovieNode와 ActorNode 두 가지가 있으므로 ActorNode인 경우 continue로 다음으로 이동하고, MovieNode인 경우만 num값을 올려주고, 영화 번호, 영화 제목, rank 점수를 출력하여준다.

(Numpy 활용)

Numpy를 이용한 방식

```
num = 0
num_RankData = {idx2node[index]:val for index, val in enumerate(rank_Numpy)}
for movie in sorted(num_RankData.items(), key=lambda x:-x[1]):
    if num == 10 :
        break
    if movie[0].split()[0] == 'movie':
        print(movie[0].split()[1],idx2movie(movie[0].split()[1]),movie[1])
        num += 1
    else:
        continue
```

```
161967 기생충 0.0015437432925532173
156464 보헤미안 랩소디 0.0010864984266341052
175322 마녀 0.0008946794759721638
174065 걸캅스 0.0008564445054703045
167651 극한직업 0.0007648489380972874
37886 클레멘타인 0.0007289295469191 59
157297 마약왕 0.0007133104346250872
71509 아저씨 0.0006938076365826392
136900 어벤져스: 엔드게임 0.0006567566198412949
163788 알라딘 0.0006387598504502 71
```

Numpy로 계산한 Rank값의 Top10 영화를 뽑은 결과이다.

위 Dictionary와 동일한 방법으로 Top10을 뽑았다.

[과제3] 영화 Top 10 노드 시각화

가장 랭크가 높은 Top 10 영화의 노드 시각화
영화와 영화의 관계는 중간에 배우가 연결 역할을 수행하여준다.
시각화를 수행하기 위해서는 첫 번째로, Top10 영화의 Node 두 번째로 그 영화의 배우 Node, 마지막으로 그 Top10 영화의 배우들이 찍은 영화들의 Node의 연결 상태로 확인이 가능하다.

(Dict를 이용한 방식)

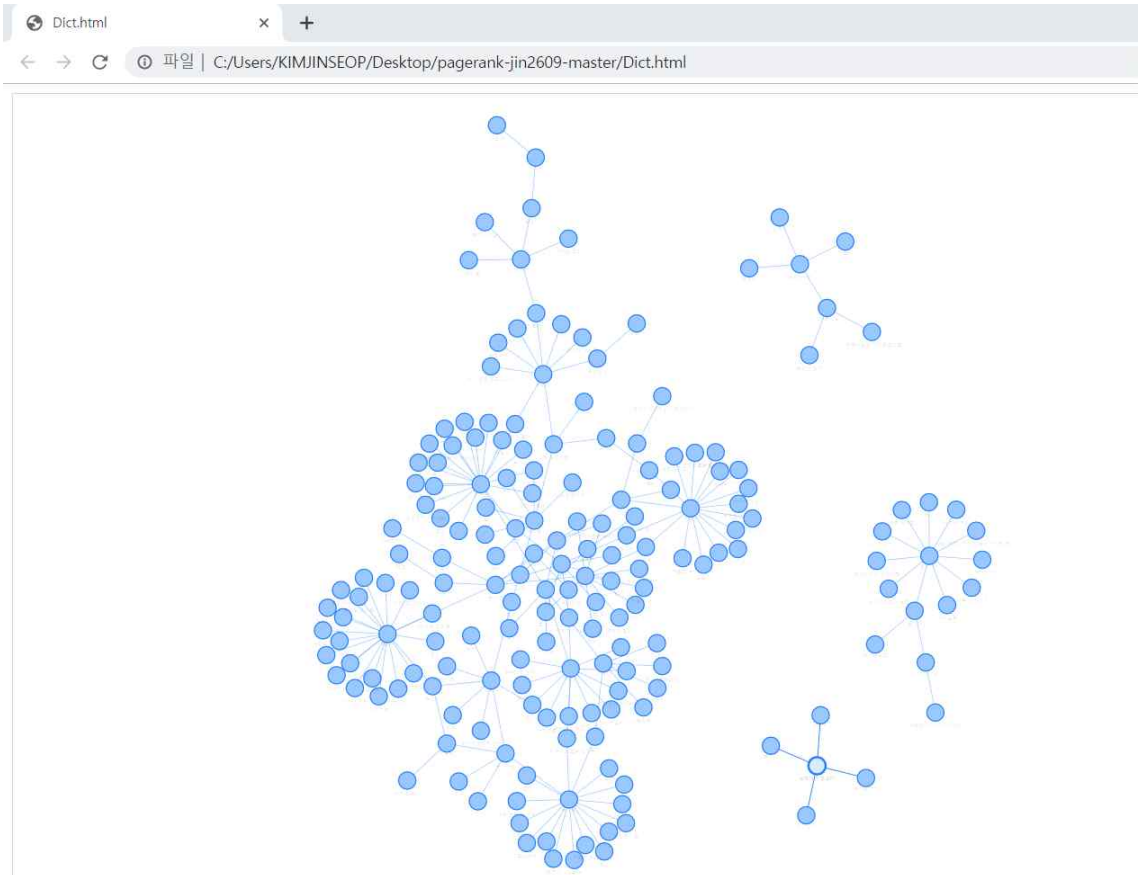
```
G_Dict=Network(1000,1000, notebook = True)
num = 0

for movie in sorted(rank_Dict.items(), key = lambda x: x[1], reverse=True):
    if num == 10 :
        break
    if movie[0].split()[0] == 'movie':
        MovieNode = idx2movie(movie[0].split()[1])
        G_Dict.add_node(MovieNode)
        for actor in g[movie[0]].items():
            ActorNode = idx2actor(actor[0].split()[1])
            G_Dict.add_node(ActorNode)
            G_Dict.add_edge(MovieNode,ActorNode)
            for actor_movie in g[actor[0]].items():
                ActorToMovieNode = idx2movie(actor_movie[0].split()[1])
                G_Dict.add_node(ActorToMovieNode)
                G_Dict.add_edge(ActorNode,ActorToMovieNode)
        num += 1
    else:
        continue

G_Dict.show('Dict.html')
```

G_Dict를 생성하여준다. notebook을 true로 하여주면, jupyter notebook에서 결과를 확인할 수 있다.
for문을 돌며 아까 계산하였던, pageRank값들을 Sort하여 Top10의 영화를 가져온다. Top10의 MovieNode를 잡았다면, 제목을 MovieNode에 저장하고, 그 것을 G_Dict의 add_node를 활용해 추가하여준다.
다음으로, 그 영화에 출연한 배우들의 정보를 가져온다. 그 정보는 g라는 그래프에 들어있으며, g에 'movie 영화번호'의 key값으로 찾아주면 그 영화에 참여한 actor들이 나오게 된다. 그 actor들을 하나하나 모두 G_Dict에 add_node로 이름을 넣어주고, 해당 영화와 배우들을 연결하여준다.
마지막으로, 그 배우들이 출연한 영화는 g에서 'actor 배우번호'를 활용하여 알아볼 수 있으며, 그 값들을 가져와서 add_node를 활용해 영화 Node를 추가하여주고, add_edge를 이용하여 그 배우와 그 배우가 출연한 영화를 연결한다.

요약하여 설명을 하자면, 3중 for문을 활용하여 위 방법을 구현하였다.
첫 번째 for문에서는 Top10의 영화를 뽑아낸다.
두 번째 for문에서는 Top10 영화에 출연한 배우들을 뽑아낸다.
세 번째 for문에서는 그 배우들이 출연한 영화들을 뽑아낸다.
그렇게 해서, 최종적으로 Top 10 노드의 시각화를 수행하였다.



Dictionary를 활용하여 만들어진 Top10을 시각화한 결과이다.

(Numpy를 이용한 방식)

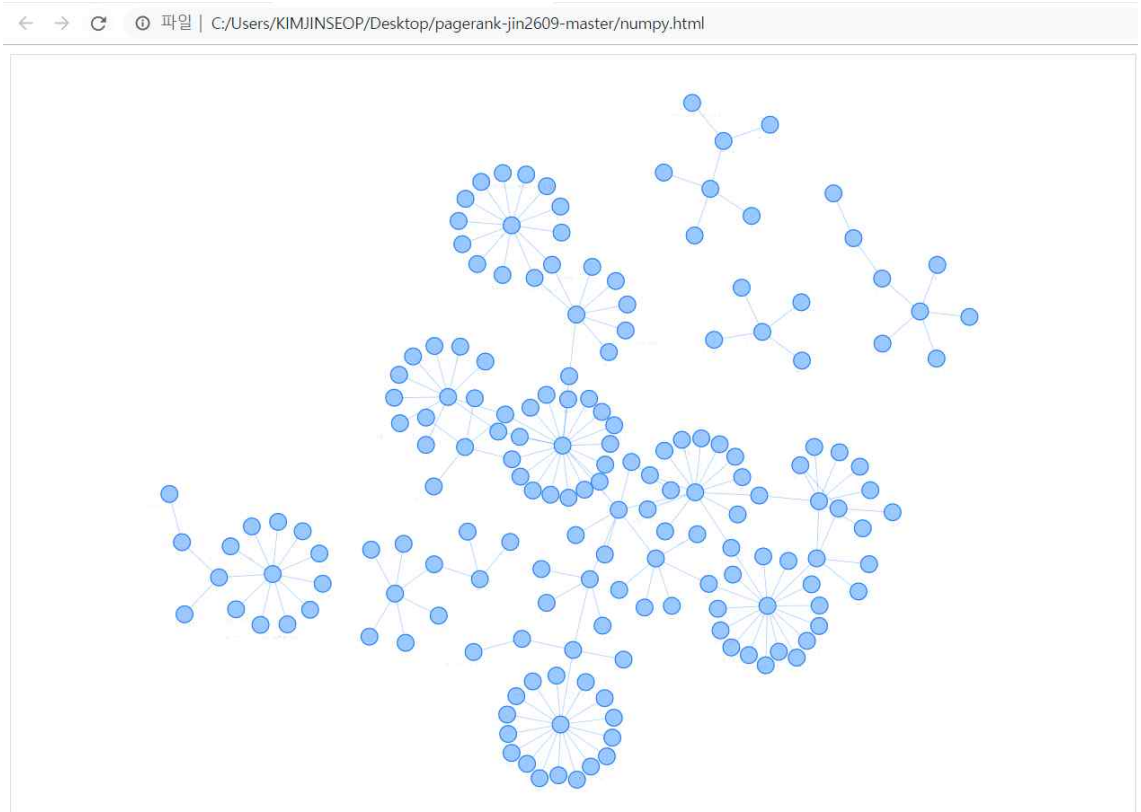
Numpy를 이용한 방식

```
G_num=Network(1000,1000, notebook = True)
num = 0

for movie in sorted(num_RankData.items(), key=lambda x:-x[1]):
    if num == 10 :
        break
    if movie[0].split()[0] == 'movie':
        MovieNode = idx2movie[movie[0].split()[1]]
        G_num.add_node(MovieNode)
        for actor in g[movie[0]].items():
            ActorNode = idx2actor[actor[0].split()[1]]
            G_num.add_node(ActorNode)
            G_num.add_edge(MovieNode,ActorNode)
            for actor_movie in g[actor[0]].items():
                ActorToMovieNode = idx2movie[actor_movie[0].split()[1]]
                G_num.add_node(ActorToMovieNode)
                G_num.add_edge(ActorNode,ActorToMovieNode)
        num += 1
    else:
        continue

G_num.show('numpy.html')
```

Numpy를 활용하는 방식도 Dictionary를 활용하는 방식과 동일하게 그래프를 그려주어 시각화를 수행한다.



Numpy를 활용하여 만들어진 Top10을 시각화한 결과이다.