Course Title (과목명)	인공지능통신
HW Number (HW 번호)	Project 4
Submit Date (제출일)	2021-06-13
Grade (학년)	4학년
ID (학번)	20151483, 20161482
Name (이름)	이창헌, 박준용

Project 4

1. 프로젝트 목표

- lecture 13의 강의자료를 참고하여 PageRank 알고리즘을 구현.

2. PageRank 알고리즘 구현

우선, PageRank 알고리즘을 구현하기 위해서는 hyperlink graph와 이로부터 도출된 matrix H에 대한 정보가 필요하다. Example slide에 제시된 8개의 node로 이루어진 graph와 이를 matrix로 정리한 데이터를 기반으로 hard-coding을 통해 matrix H를 작성한다.

```
[3]  #Matrix
     H = np.matrix('0 0.5 0.5 0 0 0 0 0; 0.5 0 0 0 0.5 0 0 0; 0 0.5 0 0 0 0 0 0.5; 0 0 1 0 0 0 0 0; 0 0 0 0.5 0 0 0 0.5; 0 0 0 0.5 0 0.5 0 0; 0 0 0 0.5 0 0.5 0 0; 0.33333333 0 0 0.33333333 0 0 0.33333333 0')
     H

     matrix([[0.        , 0.5       , 0.5       , 0.        , 0.        ,
              0.        , 0.        , 0.        ],
             [0.5       , 0.        , 0.        , 0.        , 0.5       ,
              0.        , 0.        , 0.        ],
             [0.        , 0.5       , 0.        , 0.        , 0.        ,
              0.        , 0.        , 0.5       ],
             [0.        , 0.        , 1.        , 0.        , 0.        ,
              0.        , 0.        , 0.        ],
             [0.        , 0.        , 0.        , 0.5       , 0.        ,
              0.        , 0.        , 0.5       ],
             [0.        , 0.        , 0.        , 0.5       , 0.5       ,
              0.        , 0.        , 0.        ],
             [0.        , 0.        , 0.        , 0.5       , 0.        ,
              0.5       , 0.        , 0.        ],
             [0.33333333, 0.        , 0.        , 0.33333333, 0.        ,
              0.        , 0.33333333, 0.        ]])
```

이후, mandatory score-spreading을 위한 matrix H_hat을 작성한다. 이 example의 경우, dangling node가 존재하지 않으므로 H_hat = H로 처리한 뒤, 바로 randomization을 위한 matrix G를 작성할 수 있다.

$$\mathbf{G} = \theta\hat{\mathbf{H}} + (1-\theta)\frac{1}{N}\mathbf{1}\mathbf{1}^T.$$

$$\mathbf{G} = \begin{bmatrix} 0.0188 & 0.4437 & 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.0188 \\ 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.0188 & 0.0188 & 0.0188 \\ 0.0188 & 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.4437 \\ 0.0188 & 0.0188 & 0.8688 & 0.0188 & 0.0188 & 0.0188 & 0.0188 & 0.0188 \\ 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.0188 & 0.0188 & 0.0188 & 0.4437 \\ 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.4437 & 0.0188 & 0.0188 & 0.0188 \\ 0.0188 & 0.0188 & 0.0188 & 0.4437 & 0.0188 & 0.4437 & 0.0188 & 0.0188 \\ 0.3021 & 0.0188 & 0.0188 & 0.3021 & 0.0188 & 0.0188 & 0.3021 & 0.0188 \end{bmatrix}$$

```
[5]  #Randomization
     theta = 0.85
     G = theta * H_hat + (1-theta)*(1/len(H))*np.ones((len(H),len(H)))
     G

     matrix([[0.01875   , 0.44375   , 0.44375   , 0.01875   , 0.01875   ,
              0.01875   , 0.01875   , 0.01875   ],
             [0.44375   , 0.01875   , 0.01875   , 0.01875   , 0.44375   ,
              0.01875   , 0.01875   , 0.01875   ],
             [0.01875   , 0.44375   , 0.01875   , 0.01875   , 0.01875   ,
              0.01875   , 0.01875   , 0.44375   ],
             [0.01875   , 0.01875   , 0.86875   , 0.01875   , 0.01875   ,
              0.01875   , 0.01875   , 0.01875   ],
             [0.01875   , 0.01875   , 0.01875   , 0.44375   , 0.01875   ,
              0.01875   , 0.01875   , 0.44375   ],
             [0.01875   , 0.01875   , 0.01875   , 0.44375   , 0.44375   ,
              0.01875   , 0.01875   , 0.01875   ],
             [0.01875   , 0.01875   , 0.01875   , 0.44375   , 0.01875   ,
              0.44375   , 0.01875   , 0.01875   ],
             [0.30208333, 0.01875   , 0.01875   , 0.30208333, 0.01875   ,
              0.01875   , 0.30208333, 0.01875   ]])
```

Matrix G를 구할 때는 위의 수식을 사용하였으며, weight을 0.85로 설정 후 구현한 결과 동일한 값을 보이는 것을 확인할 수 있다.

다음으로 간단한 for 문을 사용한 iteration code를 사용하여 iteration을 진행하였다.

$$\text{Initializing } \pi[0] = [1/8\ 1/8\ \cdots 1/8]^T, \text{ iteration } (3.3) \text{ gives}$$

[[0.10729167] [0.125] [0.178125] [0.21354167] [0.125] [0.071875] [0.05416667] [0.125]]	[[0.10729167] [0.14005208] [0.24585937] [0.16085937] [0.10242187] [0.04177083] [0.05416667] [0.14757812]]	[[0.12008594] [0.16883919] [0.20107943] [0.14486654] [0.09602474] [0.04177083] [0.0605638] [0.16676953]]	[[0.13775802] [0.15524528] [0.19292308] [0.1503041] [0.10825926] [0.04448962] [0.06600137] [0.14501927]]	[[0.12581804] [0.15928947] [0.20505565] [0.15280765] [0.10363733] [0.04680058] [0.05983879] [0.14675249]]
[[0.1280279] [0.15937131] [0.20210916] [0.14969747] [0.10633827] [0.04418149] [0.06032987] [0.14994451]]	[[0.12896709] [0.15905825] [0.20040471] [0.15084537] [0.10525994] [0.0443902] [0.06123428] [0.14984016]]	[[0.12880447] [0.15873301] [0.20177958] [0.15083059] [0.10521559] [0.04477457] [0.06120471] [0.14865748]]	[[0.12833115] [0.15924822] [0.2016979] [0.15062744] [0.10524072] [0.044762] [0.06086962] [0.14922295]]	[[0.12871033] [0.15901234] [0.20132406] [0.15065058] [0.10545434] [0.04461959] [0.06102983] [0.14919891]]

위의 표를 통하여 π[1] 부터 π[10] 까지의 iteration의 횟수와 그 결과를 확인할 수 있다. Iteration 결과, 값은 π[6] 정도에서 수렴하는 것을 알 수 있었다.

```
[7]  pi.round(4)

     array([[0.1287, 0.159 , 0.2013, 0.1507, 0.1055, 0.0446, 0.061 , 0.1492]])

[8]  #PageRank result
     import collections
     pg = {k: v for v, k in enumerate(pi.round(4).tolist()[0])}
     pg = collections.OrderedDict(sorted(pg.items(),reverse=True))
     pg

     OrderedDict([(0.2013, 2),
                  (0.159, 1),
                  (0.1507, 3),
                  (0.1492, 7),
                  (0.1287, 0),
                  (0.1055, 4),
                  (0.061, 6),
                  (0.0446, 5)])

[9]  print('The ranked order of the webpages are:')
     for i in range(0, len(pg.values())):
         temp = list(pg.values())[i]+1
         print(temp)

     The ranked order of the webpages are:
     3
     2
     4
     8
     1
     5
     7
     6
```

이후 Python의 정렬 기능을 사용하여 array의 순서에 따라 sorting한 다음 webpage의 ranked order를 출력할 수 있었다. 위의 과정은 **Aicomm_Project4.ipynb**에 구현되어 있다.

3. 결과 및 토의

위의 과정을 통하여 Google PageRank의 작동 원리를 알아볼 수 있었다. 구현 시 어려웠던 점은 python에서 제공하는 array와 list를 사용할 때 numpy와 혼용하는 것이 어려워 자료형을 통일하는 데에 약간의 어려움이 존재하였다. 또한, PageRank order를 출력할 때 dictionary 자료형은 sorting을 지원하지 않아, OrderedDict라는 자료형을 통하여 출력하여야 하는 어려움이 있었다.

이번 학기 인공지능통신 과목을 통하여 가장 유익하였던 주제는 아마존 movie rating system의 구현이었다. 초반에 기계학습 과정에서 배운 loss function과 parameter optimization을 막연하게 이해하고 딥러닝에 한정된 편협한 영역에서의 활용만이 가능하다고 생각하고 있었다. 그러나, 아마존 movie rating의 구현을 포함한 다양한 문제에 적용해서 차이를 줄일 수 있는 enhancement의 개념을 알게 되어, 광범위한 영역에서 error를 줄이는 방법으로 문제 해결에 접근할 수 있다는 점을 깨우칠 수 있게 되었다. 또한, 기계학습 과정에서 overfitting과 관련된 최적화 문제에 대한 접근 방법을 통하여, data를 가공할 때에 간과하기 쉬운 approximation error의 중요성에 대하여 일깨울 수 있었다.

4. References

AICom_NS2_Google_How does Google rank web pages.pdf, 서강대학교 NICE Lab.