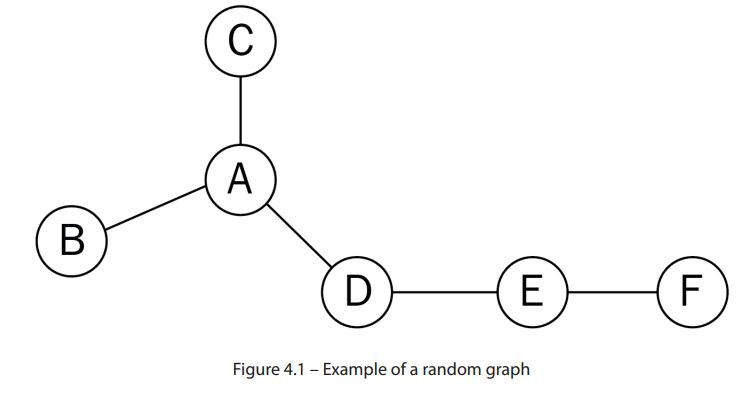
1. Node2Vec에서 편향된 랜덤워크를 통해 임베딩 향상 시키기

1) Node2Vec 소개

① 이웃(neighborhood) 정의하기



- 목적 : 노드 A 근처에 있는 3개의 노드를 탐색하고자 함. 이 과정을 ‘샘플링 전략’이라 함.

- 해결책 1) 연결 측면에서 가장 가까운(close) 세 개의 노드를 고려하는 것

이 경우, N(A)로 표시된 A의 이웃은 N(A) = {B,C,D}

즉, 노드 A에서 가장 가까운 세 개의 노드는 노드B, 노드C, 노드D

☞ BFS (너비우선탐색) 구현

- 해결책 2) 이전 노드와 인접하지 않은(not close) 노드를 먼저 선택하는 것

이 경우, N(A) = {D, E, F}

즉, 노드 A의 이웃은 노드 D, 노드 E, 노드 F

☞ DFS (깊이우선탐색) 구현

- 두 가지 네트워크 속성 (Structural equivalence & Homophily)

1) 구조적 동등성

: 노드들이 같은 이웃들을 많이 공유하면 구조적으로 동등하다 !

2) 동종선호 (자신과 비슷한 사람들과 어울리려는 경향)

: 유사한 노드끼리 연결될 가능성이 더 높다 !

**∴ 동형적이고 구조적 동등성을 결합한 그래프가 해결책 (두 알고리즘과 두 네트워크 속성을 연결)**

② 편향된 random walks 소개

cf. Random Walks는 그래프에서 무작위로 선택되는 노드의 연속(sequence)

- 무작위일 수도 있는 시작점과 미리 정의된 길이를 가짐

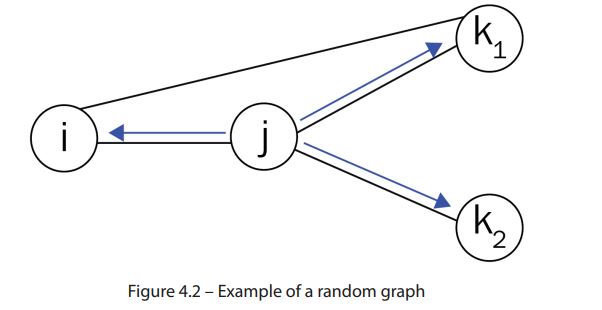
- 여기서 node는 문장에서 함께 나타나는 ‘단어’와 같음

- 동종선호 가설 하에 유사한 의미를 공유하므로 유사한 표현을 사용

- Node2Vec의 목표 : Random Walk의 무작위성을 다음 중 하나로 편향시키는 것

1) 이전 노드와 연결되지 않은 노드로 승격 (DFS와 유사)

2) 기존에 근접한 노드로 승격 (BFS와 유사)



- 현재 노드 : j , 이전 노드 : i , 미래 노드 : k

- 노드 j → 노드 k 로의 정규화되지 않은 전이확률인 에 주목

- 이 전이확률은 = 로 분해될 수 있음

- 여기서 은 노드i와 노드k 사이의 검색편향(the search bias)

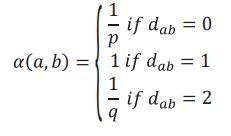
- 은 j에서 k까지의 edge의 가중치

- DeepWalk에서는 노드a와 노드b의 임의의 쌍에 대해 =1 이 있음.

- Node2Vec에서 값은 노드 사이의 거리와 두 개의 추가 매개변수(p와 in-out 매개변수 q)를 기반으로 정의

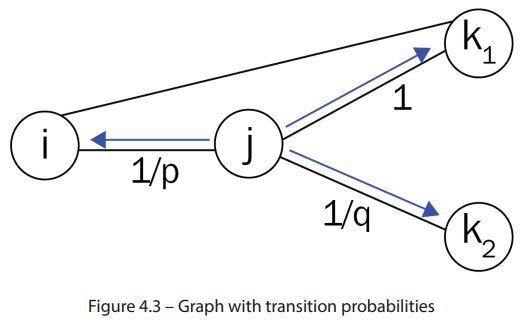
- 이들의 역할? 각각 DFS와 BFS의 근사치를 구하는 것

- 여기서 값이 정의되는 방법 :



- 는 노드a와 노드b 사이의 최단 경로거리

- 이전 그래프에서 정규화되지 않은 전이 확률을 다음와 같이 업데이트 할 수 있음 :



- 워크는 노드i에서 시작하여 노드 j에 도착

- 이전 노드i로 돌아갈 확률은 매개변수 p에 의해 제어

- 이 값이 높을수록 랜덤 워크는 동일한 노드를 반복하고 DFS처럼 보이는 대신 새로운 노드를 더 많이 탐색

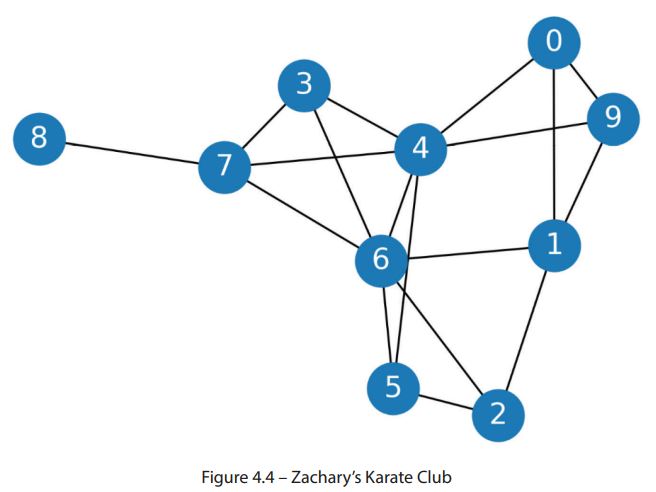
- 이전 노드인 i 바로 근처에 있기 때문에 k1으로 갈 확률은 1임

- 노드 k2로 갈 확률은 매개변수 q에 의해 제어

- 이 값이 높을수록 랜덤 워크는 이전 노드에 가깝고 BFS처럼 보이는 노드에 더 집중

☞ 이를 이해하는 가장 좋은 방법? 이 아키텍처를 실제로 구현하고 매개변수를 가지고 노는 것

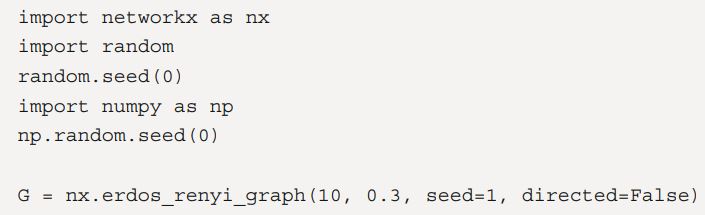
③ (그림 4.4) 재커리의 가라테 클럽에서 단계별로 구현하기



- 가중치가 없는 네트워크이므로 전환확률(the transition probability)은 검색 편향에 의해서만 결정됨

☞ 먼저 이전 노드와 현재 노드, 그리고 두 개의 파라미터 p와 q를 기반으로 그래프에서 다음 노드를 무작위로 선택할 함수를 만들고자 함.

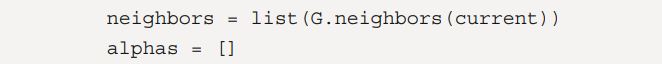
⑴ 필요한 라이브러리 생성



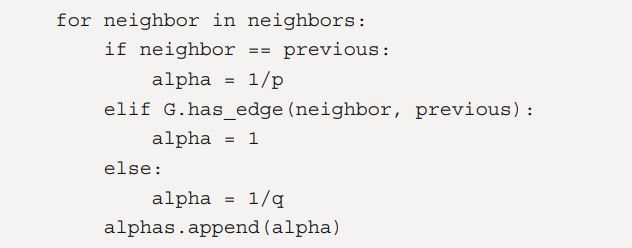
⑵ 매개변수 목록으로 next\_node 함수를 정의



⑶ 현재 노드에서 이웃 노드 목록을 검색하고 알파 값 목록을 초기화



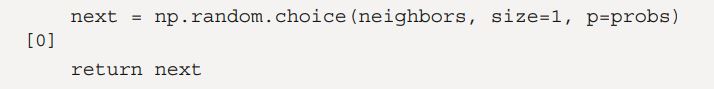
⑷ 각 이웃에 대해 적절한 알파 값을 계산 (이웃이 이전 노드라면 1/p, 이웃이 이전 노드에 연결되어 있으면 1, 그렇지 않으면 1/q)



⑸ 이 값들을 정규화하여 확률을 생성



⑹ np.random.choice()를 사용하여 이전 단계에서 계산된 전이 확률을 기반으로 다음 노드를 무작위로 선택하고 반환



- 이 기능을 테스트하기 전 전체 랜덤워크를 생성하기 위한 코드 필요

- 이러한 랜덤워크를 생성하는 방법 ? 이전 장에서 보았던 것과 유사.

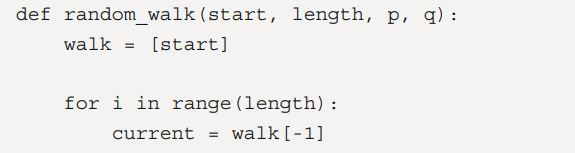
- 차이점 : 다음 노드가 next\_node() 함수에 의해 선택된다 !

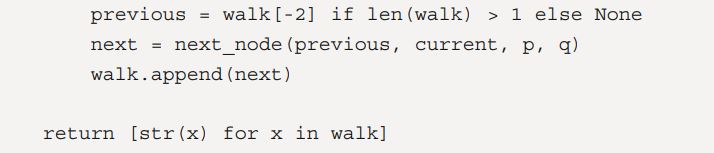
- next\_node() 함수는 이전 노드와 현재 노드 뿐만 아니라 추가적인 매개변수를 필요

- 이 노드들은 walk 변수에 추가된 마지막 두 개의 요소를 보면 쉽게 얻을 수 있음

- 호환성의 이유로 정수 대신 문자열을 반환하기도 함

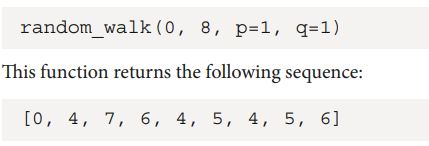
- 새로운 버전의 random\_walk() 함수





→ random walk를 생성할 수 있는 모든 요소가 갖춰짐 !

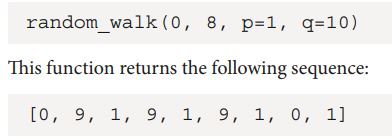
→ 이제 길이가 5, p=1, q=1인 것을 시도해보겠다.

⑴ 

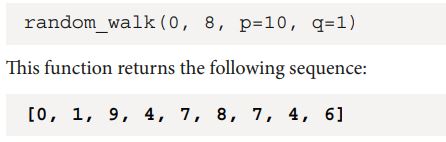
- 모든 이웃 노드가 동일한 전이확률을 가지므로 random이어야 함.

- 이러한 매개변수를 사용하여 정확한 DeepWalk 알고리즘을 재현한다.

⑵ q=10을 사용하여 이전 노드로 돌아가는 방향으로 편향시키기



⑶ 랜덤워크가 그래프에서 더 많은 노드를 탐색하는 예. p=10으로 확률이 낮기 때문이 이전노드로 돌아가지 않음



☞ 실제 사례에서 이러한 속성을 사용하는 방법을 살펴보고 DeepWalk와 비교

2) Node2Vec 구현

- 이제 편향된 랜덤워크를 생성할 수 있는 기능을 갖게 되었으므로 Node2Vec의 구현은 DeepWalk를 구현하는 것과 매우 유사

- 동일한 코드를 재사용하고 p=1 및 q=1로 시퀀스를 생성하여 DeepWalk를 Node2Vec의 특수한 경우로 구현할 수 있다는 점이 매우 유사

- 이 작업을 위해 재커리의 가라테 클럽을 다시 사용 !

- 이전 장에서와 같이 클럽의 각 구성원을 두 그룹(”Mr.Hi” 및 “Officer”) 중 하나의 일부로 올바르게 분류하는 것이 목표

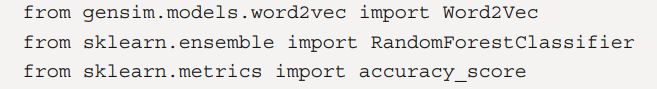
- Node2Vec에서 제공하는 노드 임베딩을 기계 학습 분류기(이 경우 랜덤 포레스트)의 입력으로 사용

- 구현 방법을 단계별로 살펴보기 :

① 먼저, Word2Vec을 사용하기 위해 gensim 라이브러리를 설치. 호환성을 이유로 버전 3.8.0을 사용



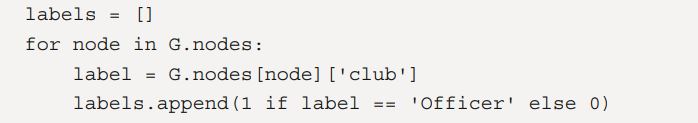
② 필요한 라이브러리 가져오기



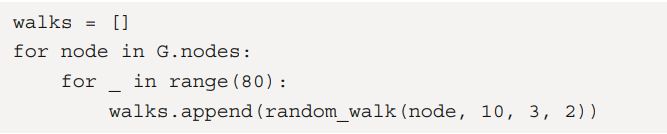
③ 데이터셋 로드하기 (자카리의 가라테 클럽)



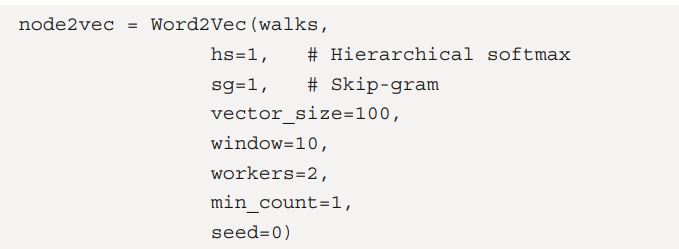
④ 노드의 레이블을 숫자 값(0 및 1)으로 변환



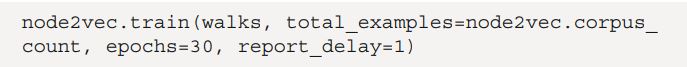
⑤ 그래프의 각 노드에 대해 random\_walk() 함수를 80회 사용하여 이전에 본 것처럼 랜덤 워크 목록을 생성한다. 그리고 여기에 지정된 것과 같은 매개 변수는 각각 2와 1임



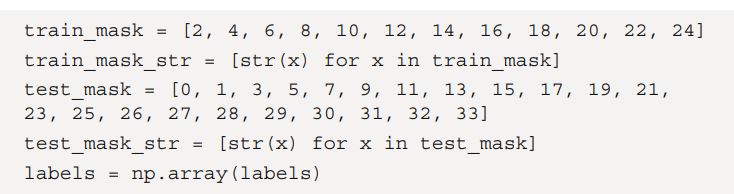
⑥ 계층적 소프트맥스 함수를 사용하여 Word2Vec(skip-gram model)의 인스턴스를 생성



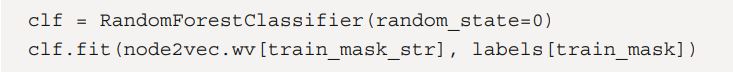
⑦ skip-gram 모델은 30개의 에포크(epochs) 동안 생성된 시퀀스(연속)에 대해 훈련됨



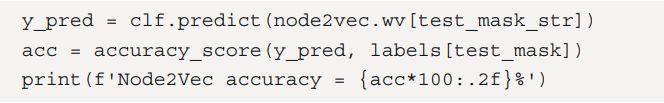
⑧ 분류기를 훈련하고 테스트를 하기 위해 mask 생성

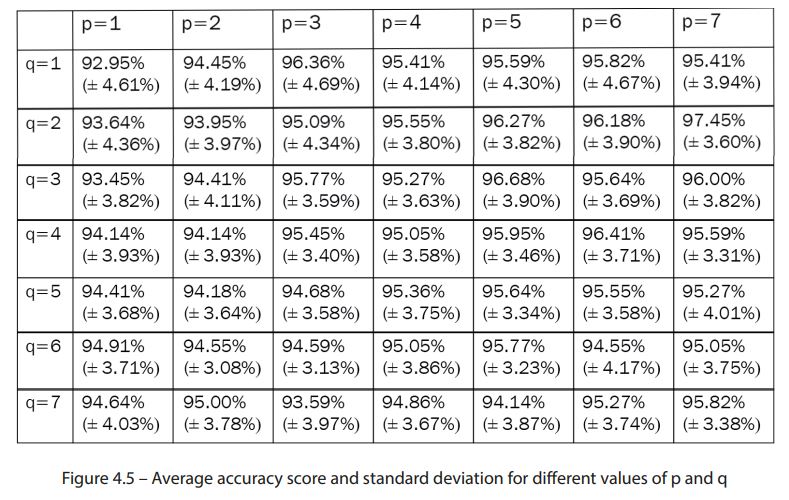


⑨ random forest 분류기는 훈련 데이터에 대해 다음과 같이 훈련됨



⑩ 테스트 데이터에 대한 정확도 측면에 대해 평가하기





- 결과를 요약한 표 (p와 q의 서로 다른 값에 대한 평균 정확도 점수 및 표준편차)

- Deepwalk를 구현하기 위해 p=1과 q=1로 정확히 동일한 과정을 반복할 수 있음

- 서로 다른 매개변수 설정으로 실험을 수행할 때, 단일한 정확도 점수만으로 모델들을 비교할 때의 문제 → 왜냐하면 같은 모델이라도 여러 번 실행할 때마다 다른 결과를 얻을 수 있기 때문에, 단일한 정확도 점수만으로는 모델들을 공정하게 비교하기 어려움

- 결과의 무작위성을 제한하기 위해, 이 과정을 100번 반복하고 평균값을 취함

- 이 결과는 훨씬 더 안정적이고 정확도 점수의 변동성을 측정하기 위해 표준편차 (np.std())를 포함할 수 도 있음

- 하지만 이것을 하기 전 게임을 해보겠다.

- 이전 장에서 자카리의 가라테 클럽 대해 동종(homophlic) 네트워크로 이야기 함

- DFS가 강조한 이 속성은 매개변수 p를 증가시키면 더 향상됨.

- DFS와 동종 네트워크의 연관성이 사실이라면, 더 높은 p 값으로 더 좋은 결과를 얻어야 함 !

- p와 q 값에 대해 1에서 7 사이의 동일한 실험을 반복 !

- 실제 머신러닝 프로젝트에서 우리는 이 매개변수 검색을 수행하기 위해 유효성 검증 데이터를 사용함.

- 이 예제에서는 이 연구가 이미 우리의 최종 응용 프로그램이기 때문에 테스트 데이터를 사용함.

☞ 몇 가지 눈에 띄는 결과

① DeepWalk(p =1 및 q =1)은 여기에서 다루는 p와 q의 다른 조합보다 성능이 떨어짐

- 이는 이 데이터 세트에 대해서도 해당됨 !

- 편향된 랜덤워크가 얼마나 유용할 수 있는지를 보여줌.

- 그러나 항상 편향된 랜덤워크가 유용한 것은 X

- 편향되지 않은 랜덤 워크는 다른 데이터 셋에서도 더 나은 성능을 발휘할 수 O

② p의 값이 높으면 성능이 향상됨 (이는 가설을 증명함)

- 이것이 소셜 네트워크라는 것을 아는 것은 랜덤 워크를 homophily (동질적인) 방향으로 편향시키는 것이 좋은 전략임을 시사

- 이런 종류의 그래프를 다룰 때 유의해야 할 점임 !

③ 매개변수를 자유롭게 바꾸면서 다른 흥미로운 결과를 찾도록 해라.

- p(>7)의 값이 매우 높거나 반대로 p와 q의 값이 0에서 1 사이인 결과를 탐색할 수 있었음 !

∴ 재커리의 가라테 클럽은 기본 데이터 셋이지만, 이 기술을 사용하여 훨씬 더 흥미로운 애플리케이션을 구축하는 방법은 다음 섹션에서 살펴본다.

3) 영화 추천시스템 구축

- GNN에서 가장 인기 있는 응용 프로그램 중 하나는 RecSys(추천시스템)

- 만약 Word2Vec (그리고, DeepWalk와 Node2Vec)의 기초에 대해 생각해보자면, 목표는 그것들의 유사성을 측정할 수 있는 벡터를 만드는 것 !

- 단어 대신에 영화를 인코딩하면, 갑자기 주어진 입력 제목과 가장 비슷한 영화를 요청할 수 있음 → 이는 RecSys와 많이 비슷

- 하지만 영화를 인코딩하는 방법은 ?

- 영화의 (편향된) 랜덤워크를 만들고 싶지만, 이를 위해서는 비슷한 영화들이 서로 연결된 그래프 데이터셋이 필요 !

- 이는 찾기가 쉽지 않음.

- 또 다른 방법은?

- 사용자 평점을 살펴보는 것 !

- 평점을 기반으로 그래프를 구축하는 다양한 기술이 있다 :

- 양측 그래프 (bipartite graphs)

- 점별 상호 정보 (pointwise mutual information)를 기반으로 하는 edge 등

- 이 절에서는 쉽고 직관적인 방법, 즉 동일한 사용자가 좋아하는 영화를 연결하는 방법을 소개함

- 그런 다음 Node2Vec을 사용한 영화 임베딩을 학습하기 위해 이 그래프를 사용할 것

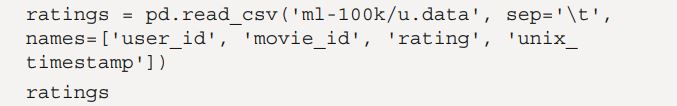
① 먼저 데이터 셋을 다운로드 함. MovieLens[2]는 100,836개의 시청률, 9,742개의 영화 및 610명의 사용자로 구성된 최신 데이터 셋(09/2018)의 작은 버전으로 인기있는 선택임. 다음과 같은 파이썬 코드로 다운로드가 가능.



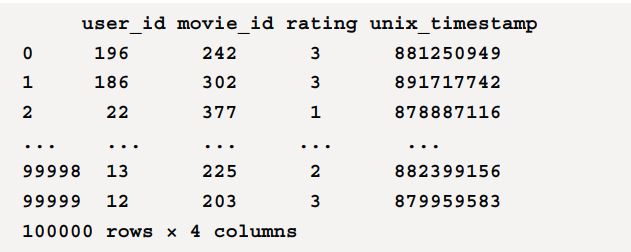
② ratings.csv와 movie.csv의 두 파일에 관심이 있음. ratings.csv는 사용자가 만든 모든 평점을 저장하고, movie.csv는 영화 식벽자를 제목으로 변환할 수 있게 해줌

③ pd.read\_csv()를 사용하여 pandas와 함께 가져오면 확인이 가능

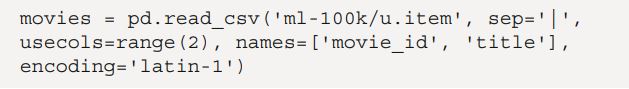




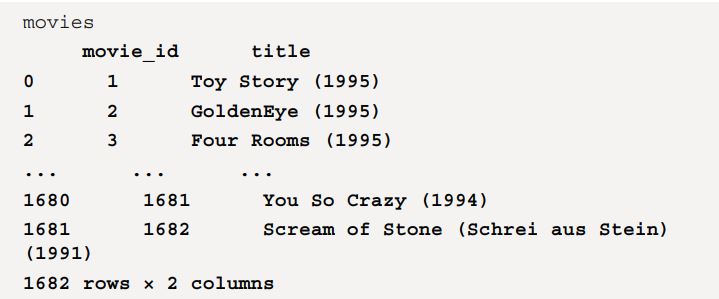
④ 이를 통해 다음과 같은 출력을 얻을 수 있음



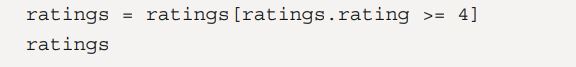
⑤ 이제 movie.csv를 가져오기



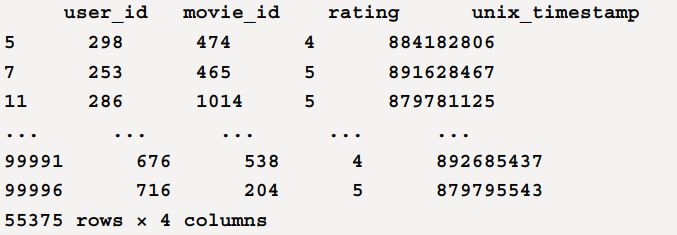
⑥ 이 데이터 셋은 다음과 같은 출력을 제공함 (movie.csv)



⑦ 여기서 동일한 사용자가 좋아하는 영화를 보고 싶으므로, 4점과 5점을 받은 평가만을 유의미한 정보로 간주하고, 다른 낮음 점수들은 무시

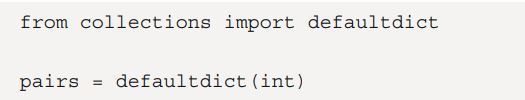


⑧ 이를 통해 다음과 같은 출력을 얻음



⑨ 현재 610명의 사용자가 만든 평점은 48,580점. 다음 단계는 동일한 사용자가 두 편의 영화를 좋아하는 횟수를 매번 세는 것임. 데이터 셋의 모든 사용자에 대해 이 과정을 반복함.

⑩ 문제를 단순화하기 위해 누락된 항목이 자동으로 생성되는 defaultdict 데이터 구조를 사용. 이를 사용하여 좋아하는 영화를 함께 계산함.



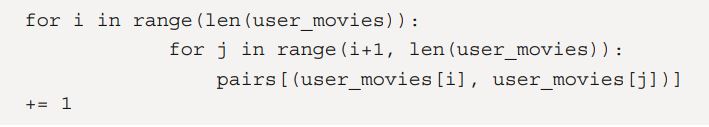
⑪ 데이터 셋의 전체 사용자 목록을 살펴보기



⑫ 현재 사용자가 좋아하는 영화 목록을 검색하기



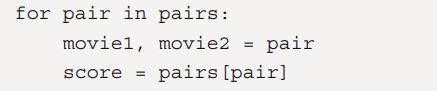
⑬ 한 쌍의 영화가 같은 목록에 함께 표시될 때마다 특정 카운터를 늘림



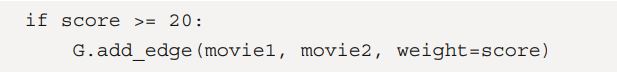
⑭ 쌍 객체는 동일한 사용자가 두 영화를 좋아한 횟수를 저장함. 이 정보를 사용하여 다음과 같이 그래프의 edge를 만들 수 있음.

⑮ networkx 라이브러리를 사용하여 그래프를 생성



⑯ 쌍 구조에서 영화의 각 쌍에 대해 두 영화와 해당 점수를 unpack 함

⑰ 이 점수가 10점보다 높으면 이 점수를 기반으로 두 영화를 연결하기 위해 그래프에 가중치 링크를 추가함. 10점보다 낮으면 연결의 의미가 떨어지는 큰 그래프가 생성되므로 10점보다 낮은 점수는 고려하지 않음.



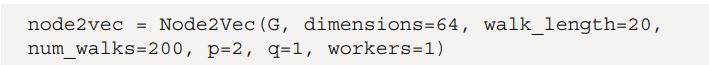
⑱ 우리가 만든 그래프에는 410개의 노드(영화)와 14,936개의 edge가 있음. 이제 노드 임베딩을 학습하기 위해 node2vec을 훈련시킬 수 있음 !

☞ 실제로 Node2Vec에 전적으로 특화된 파이썬 라이브러리가 있으므로 다음 예제에서 그것을 시도해보기로 함.

① Node2Vec 라이브러리를 설치하고 Node2Vec 클래스를 가져오기



② p 및 q 매개변수를 기반으로 편향된 랜덤워크를 자동으로 생성하는 Node2Vec 인스턴스 생성하기

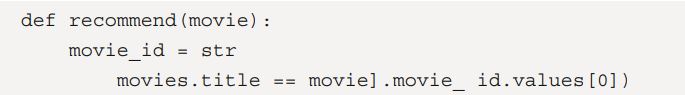


③ 다음과 같이 10개 (이전에는 5개, 이후에는 5개)의 창으로 이러한 편향된 랜덤워크에 대한 모델을 훈련하기

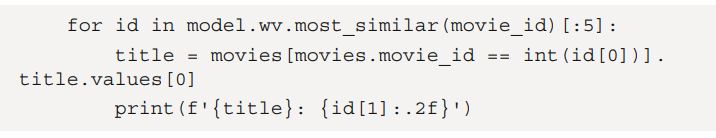


Node2Vec 모델은 훈련되어 있으며 이제 gensim 라이브러리에서 Word2Vec 개체를 사용하는 것과 동일하기 사용할 수 있음. 주어진 제목을 기반으로 영화를 추천하는 함수를 만들어보자.

④ 영화 제목을 입력으로 받은 recommend() 함수를 만든다. 이 함수는 제목을 영화 ID로 변환하여 모델을 쿼리하는 데 사용할 수 있음



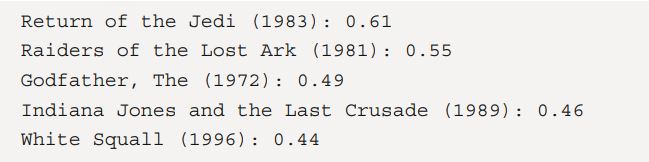
⑤ 가장 유사한 다섯 개의 단어 벡터를 순환함. 이 ID를 해당 유사성 점수로 인쇄한 영화 제목으로 변환함.



⑥ 코사인 유사도에서 스타워즈와 가장 유사한 5편의 영화를 얻기 위해 이 함수를 호출함.



⑦ 다음과 같은 출력을 받는다.



☞ 이 모델은 Jedi의 Return과 Lost Ark의 Raiders가 상대적으로 낮은 점수 (<0.7)을 얻었지만 스타워즈와 유사하는 것을 알려줌 !

☞ 그럼에도 불구하고, 이것은 RecSys 세계로의 첫 걸음을 내딛는 좋은 결과임. 이후 장에서, 우리는 최첨단 RecSys를 구축하기 위한 더 강력한 모델과 접근 방식을 볼 것 !

∴ 요약

- 이 장에서는 Word2Vec을 기반으로 한 두 번째 아키텍처인 Node2Vec에 대해서 배움.

현향된 랜덤워크를 생성하는 함수를 구현하고, 이들의 매개변수와 두 가지 네트워크 속성인 동질성(homophily)과 구조적 동등성(structural equivalence) 사이의 연결을 설명함.

- 자카리의 가라테 클럽에 대한 Node2Vec의 결과를 DeepWalk의 결과와 비교하여 유용성을 보임.

- 마지막으로, 사용자 정의 그래프 데이터셋과 Node2Vec의 다른 구현을 사용하여 첫 번째 RecSys을 구축하였음.

☞ 다음 장에서는 ?

- 바닐라 신경망을 사용하여 Node2Vec와 DeepWalk에서 자주 무시되는 문제 중 하나에 대해 이야기 : 적절한 노드 특성의 부재

- 이 문제를 네트워크 topology를 이해할 수 없는 전통적인 신경망을 사용하여 해결하려고 시도

**5장. Including Node Features with Vanilla Neural Networks**

- 지금까지 고려한 정보 유형은 그래프의 토폴로지(topology) 뿐임.

- 그러나 그래프 데이터셋은 단순한 연결의 집합보다 더 풍부할 수 있음 !

- 노드와 edge는 점수, 색상, 단어 등을 나타내는 기능(feature)을 가질 수 O

- 이 추가 정보를 입력 데이터에 포함하는 것은 최상의 임베딩을 생성하는 데 필수적

- 사실, 이는 머신러닝에서 자연스러운 일임.

- 노드와 edge 기능의 표 형식은 (비 그래프) 데이터 셋과 동일한 구조를 가짐

- 이는 전통적인 기술이 이 데이터에 적용될 수 있다는 것을 의미

- 이러한 추가 정보를 고려한 노드 표현을 포함하는 것은 모든 GNN 아키텍처의 기초

- 이를 통해 모델의 성능을 크게 향상시킬 수 있음

- 이번 장에서는 다음과 같은 주제를 다룸

**1) 그래프 데이터셋 소개**

**2) 바닐라 신경망을 사용한 노드 분류**

**3) 바닐라 그래프 신경망을 사용한 노드 분류**

---

**1) 그래프 데이터셋 소개**

- 이 장에서 사용할 그래프 데이터셋은 자카리의 가라테 클럽보다 더 풍부

- 더 많은 노드, 더 많은 edge가 있으며, 노드 기능을 포함한다.

- 이 섹션에서는 PyTorch Geometric을 사용하여 이러한 그래프를 처리하는 방법을 이해하기 위해 이들을 소개함.

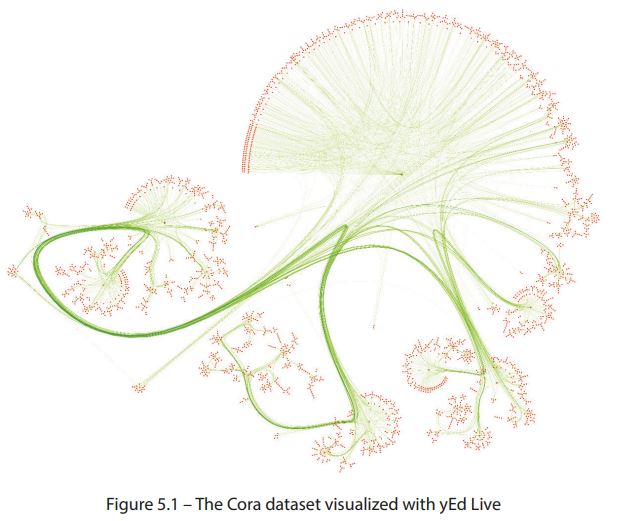
- 사용할 두 개의 데이터셋은 다음과 같음 :

① Cora 데이터셋

② Facebook Page-Page 데이터셋

---

① Cora 데이터셋



- 그림 5.1은 yEd Live로 생성된 Cora 데이터셋의 플룻

- 주황색으로 표시된 노드 : 논문

- 녹색으로 표시된 edge : 그 사이의 연결

- 일부 논문(node)은 서로 교차되어 군집을 형성함.

- 이러한 군집은 연결이 적은 노드보다 분류하기 쉬울 것으로 예상

- 학술 노드 분류를 위한 가장 인기있는 데이터셋 중 하나

- 2,708개의 출판물 네트워크를 나타내며, 각 연결은 참조를 나타냄.

- 각 출판물은 1,433개의 고유한 단어로 이루어진 이진 벡터로 설명되며, 해당 단어의 부재 또는 존재를 나타내는 0과 1로 표시됨. 이 표현은 자연어 처리에서 이진 단어 가방 (binary bag of words)라고도 함.

- 목표 : 각 노드를 7개의 범주 중 하나로 분류하는 것 !

- 그래프는 networkx와 같은 Python 라이브러리를 사용하여 시각화하기에 너무 커질 수 있으므로 그래프 데이터 시각화 전용 도구를 사용함 !

- 가장 인기있는 두 가지 도구를 본 책에서는 사용한다.

- yEd Live

- Gephi

- 이를 가져와서 PyTorch Geometric으로 주요 특성을 분석해보자.

- 이 라이브러리에는 데이터셋을 다운로드하고 관련 데이터 구조를 반환하는 전용 클래스가 있음. 여기서 PyTorch Geometric은 이미 설치되어 있다고 가정함 :

⑴ PyTorch Geometric에서 Planetoid 클래스를 가져옴



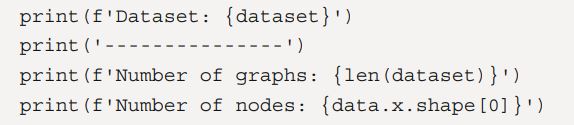
⑵ 이 클래스를 사용하여 데이터셋을 다운로드 함

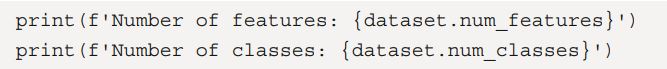


⑶ Cora는 하나의 그래프만 포함하므로 이를 전용 데이터 변수에 저장할 수 있음

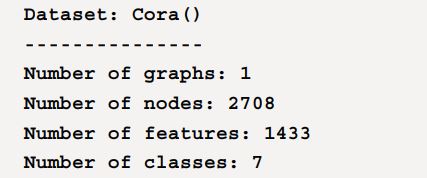


⑷ 데이터셋에 대한 일반적인 정보를 출력해보자

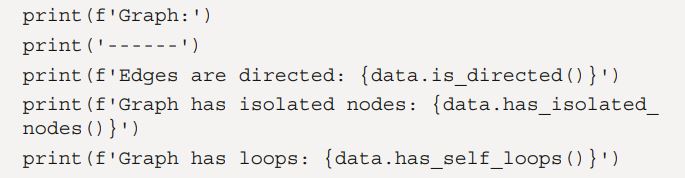




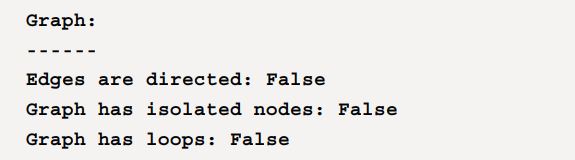
⑸ 이렇게 하면 다음과 같은 출력이 나옴



⑹ PyTorch Geometric의 전용 함수를 사용하여 자세한 정보를 얻을 수도 있음



⑺ 이렇게 하면 다음과 같은 결과가 나옴



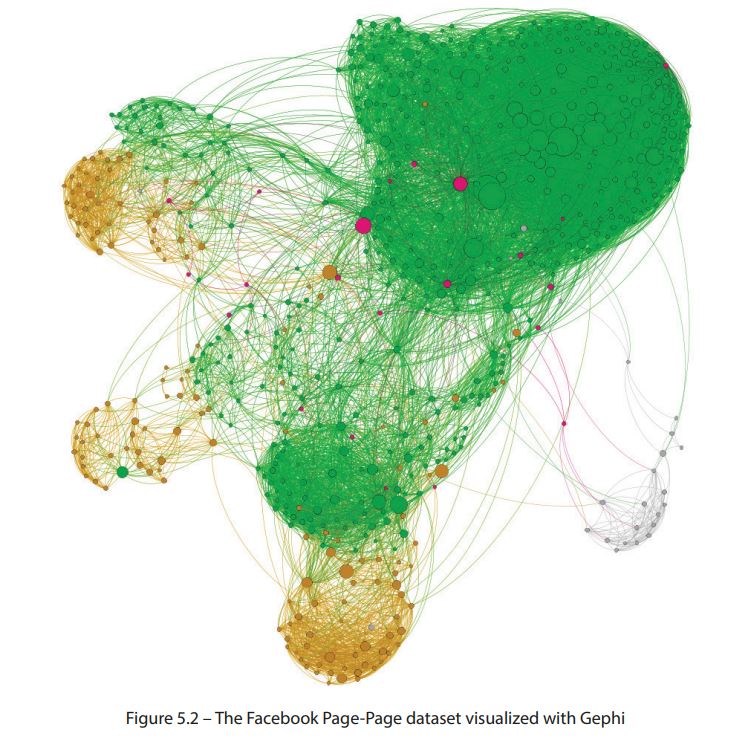
☞ 첫 번째 출력은 노드, 기능(features), 클래스 수에 대한 정보를 확인

☞ 두 번째 출력은 그래프 자체에 대한 더 많은 통찰력을 제공:

- edge는 무방향이며, 각 노드에는 이웃이 있으며, 그래프에는 self-loop가 없음

- PyTorch Geometric의 utils 함수를 사용하여 다른 속성을 테스트할 수 있지만, 이 예에서는 새로운 정보를 배우지 않음

② Facebook Page-Page 데이터셋 (실제 세계의 소셜 네트워크의 규모를 대표하는 더 큰 데이터셋)



- 그림 5.2는 Gephi를 사용하여 데이터셋을 시각화한 것

- 먼저, 연결이 적은 노드가 필터링되어 성능을 향상시킴

- 남은 노드의 크기는 연결 수에 따라 다르며, 색상은 그들이 속한 카테고리를 나타냄

- 마지막으로, Fruchterman-Reingold 및 ForceAtlas2 두 가지 레이아웃이 적용

- Facebook Page-Page 데이터셋을 Cora와 같은 방식으로 가져올 수 있음

- 이 데이터셋에서 22,470개의 노드 각각은 공식 Facebook 페이지를 나타냄

- 페이지는 서로가 서로를 좋아하는 경우에 연결

- 노드 특성(128차원 벡터)은 이러한 페이지의 소유자가 작성한 텍스트 설명에서 생성

- 목표 : 각 노드를 정치인, 회사, 텔레비전 프로그램 및 정부 기관 중 하나로 분류하기 !

- Cora와 세 가지 주요 차이점

⑴ 노드 수가 훨씬 많음 (2,708 : 22,470)

⑵ 노드 특성의 차원이 급격히 감소 (1,433 → 128)

⑶ 각 노드를 7개가 아닌 4가지 범주로 분류하는 것이 목표 (옵션이 적으므로 더 쉬움)

---

⑴ PyTorch Geometric에서 FacebookPagePage 클래스를 가져오기



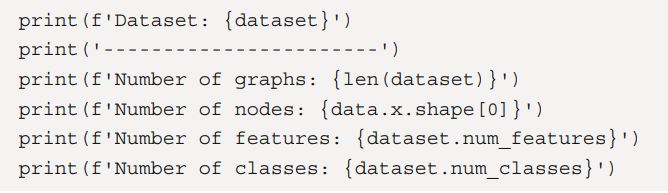
⑵ 이 클래스를 사용하여 데이터셋을 다운로드 하기



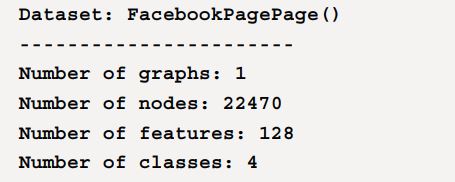
⑶ 전용 데이터 변수에 그래프를 저장하기



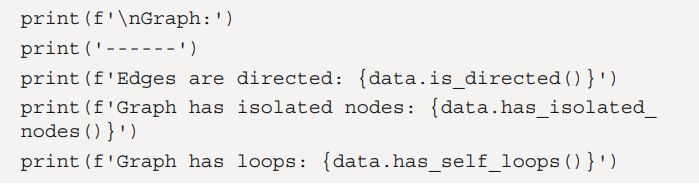
⑷ 데이터셋에 대한 일반적인 정보를 출력해보기



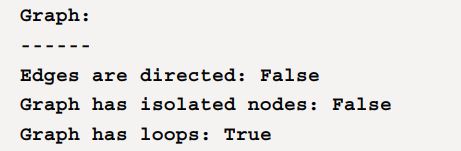
⑸ 이렇게 하면 다음과 같이 출력이 나옴



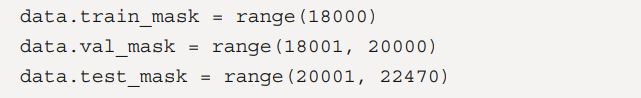
⑹ 동일한 전용 함수를 여기에도 적용할 수 있음



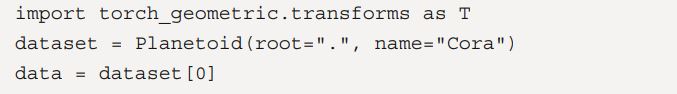
- 이렇게 하면 다음과 같은 결과가 나옴



⑺ Cora와 달리, Facebook Page-Page에는 기본적으로 학습, 평가 및 테스트 마스크가 없음. range() 함수를 사용하여 임의의 마스크를 만듦



또는 PyTorch Geometric은 데이터셋이 로드될 때 무작위 마스크를 계산하기 위해 transform 함수를 제공함



☞ 첫 번째 출력 : 데이터 셋 설명에서 본 노드 및 클래스 수를 확인

☞ 두 번째 출력 : 이 그래프에 self loop가 있다는 것을 알려줌

- 즉, 일부 페이지가 자신과 연결되어 있음 !

∴ 다음 섹션에서 첫 번째 GNN 성능과 바닐라 신경망 성능을 비교하기 위해 이 두 그래프 데이터셋을 사용할 것임 !!

**2) 바닐라 신경망을 사용한 노드 분류**

- 바닐라 신경망을 사용하여 노드를 분류할 때, 자카리의 가라테 클럽과 비교해서 이 두 데이터셋에는 새로운 유형의 정보가 포함되어 있음: 노드 특성

- 이것들은 소셜 네트워크에서 사용자의 나이, 성별, 관심사 등과 같이 그래프의 노드에 대한 추가 정보를 제공

- 바닐라 신경망 (= 다층 퍼셉트론)에서는 이러한 임베딩이 직접 모델에 사용되어 노드 분류와 같은 하류 작업을 수행함

- 이 섹션에서는 노드 특성을 일반적인 표 형식의 데이터셋으로 간주

- 이 데이터셋에서 간단한 신경망을 훈련하여 노드를 분류함

- 이 아키텍처는 네트워크의 토폴로지를 고려하지 X

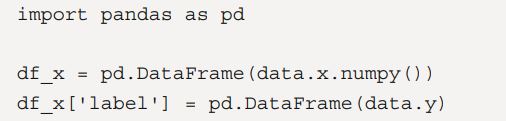
- 다음 섹션에서 이 문제를 해결하고 결과를 비교 !

- 노드 특성의 표 형식 데이터셋은 생성한 데이터 객체를 통해 쉽게 접근이 가능

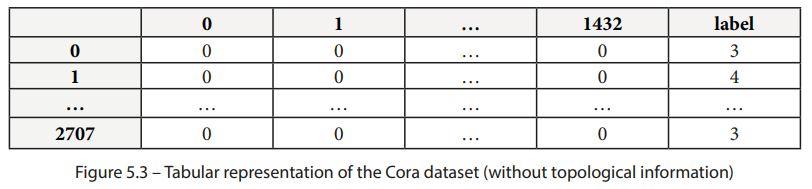
- 먼저, 이 개체를 일반적인 pandas DataFrame으로 변환

- 이를 위해 data.x(노드 특성을 포함하는)와 data.y(각 노드의 클래스 레이블을 포함하는)을 병합

- 다음은 Cora 데이터셋을 사용하는 예시



- 이를 통해 다음과 같은 데이터셋을 얻을 수 있음



- 만약 머신러닝에 익숙하다면, 데이터와 레이블이 포함된 전형적인 데이터셋을 알아볼 수 있을 것임. 우리는 data.x에서 제공된 레이블로 간단한 다층 퍼셉트론(MLP)을 개발하고 훈련시킬 수 있음 !

- 다음과 같이 4가지 메서드를 가진 자체 MLP 클래스를 만들어 보자.

- \_\_init\_\_() : 인스턴스를 초기화

- forward() : forward pass를 수행

- fit() : 모델을 훈련

- test() : 모델을 테스트

- 모델을 훈련하기 전, 주요 메트릭을 정의해야 함 !

- 다중 클래스 분류 문제에는 여러가지 메트릭이 있음 :

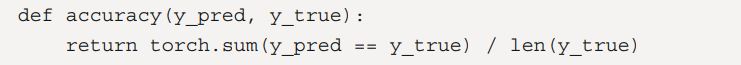
- 정확도, F1 점수, ROC AUC 점수 등

- 이 작업에서는 간단한 정확도를 구현해보겠음.

- 정확도는 올바른 예측의 비율로 정의됨.

- 다중 클래스 분류에 대한 최상의 메트릭은 아니지만 이해하기 쉬움.

(원하는 메트릭으로 바꿔서 사용해도 됨 !)

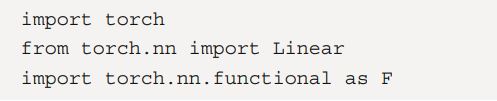


☞ 이제 실제 구현을 시작 !

- 이 섹션에서는 MLP를 구현하는 데 PyTorch Geometric이 필요하지 X

- 모든 작업은 정규 PyTorch로 수행 O

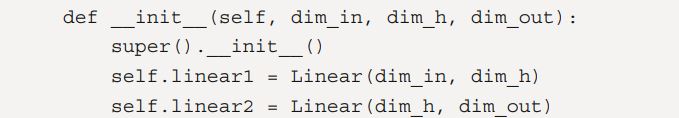
⑴ PyTorch에서 필요한 클래스 가져오기



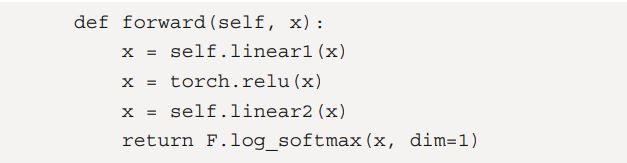
⑵ torch.nn.Module의 모든 메서드와 속성을 상속하는 MLP라는 새 클래스 만들기



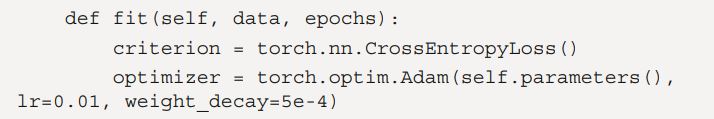
⑶ \_\_init\_\_() 메서드에는 입력, 은닉 및 출력 레이어의 뉴런 수에 대한 세 가지 인수 (dim\_in, dim\_h, dim\_out)가 있음. 또한, 두 개의 선형 레이어를 정의함.



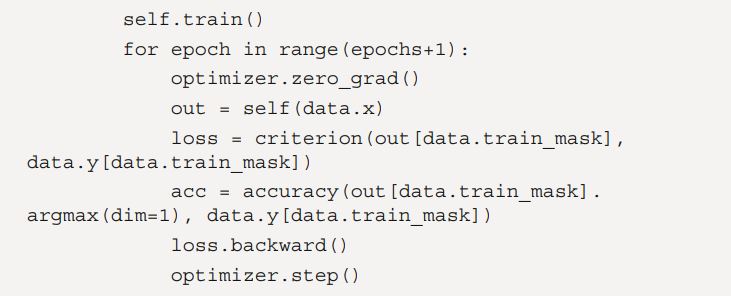
⑷ forward() 메서드는 forward pass를 수행함. 입력은 첫 번째 선형 레이어에 ReLU(활성화 함수)를 사용하여 전달되며, 결과는 두 번째 선형 레이어로 전달됨. 분류를 위해 최종 결과의 로그 소프트맥스를 반환함.



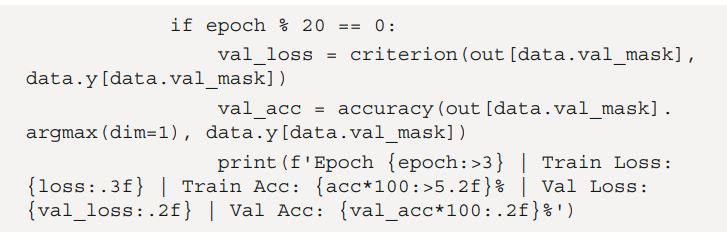
⑸ fit() 메서드는 훈련 루프를 담당함. 먼저, 훈련 과정에서 사용될 손실 함수와 옵티마이저를 초기화 함.



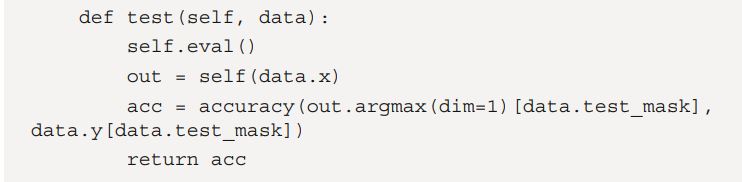
⑹ 그런 다음 일반적인 PyTorch 훈련 루프를 구현함. 손실 함수 위에 accuracy() 함수를 사용함.



⑺ 같은 루프에서 매 20번째 에포크마다 훈련 및 평가 데이터에 대한 손실과 정확도를 그림.



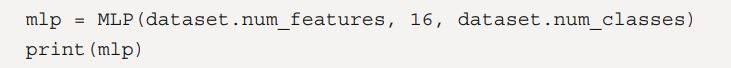
⑻ test() 메서드는 모델을 테스트 세트에서 평가하고 정확도 점수를 반환함.



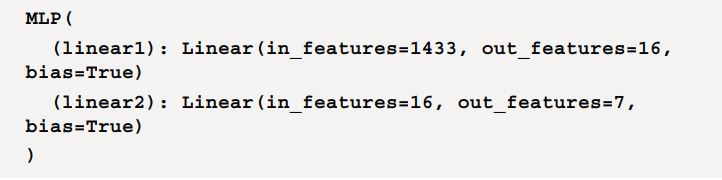
→ 이제 클래스가 완성되었으므로 MLP의 인스턴스를 만들고 훈련 및 테스트가 가능 !

☞ 두 개의 데이터셋이 있으므로, Cora에 대한 모델과 Facebook Page-Page에 대한 모델이 필요함. 먼저, Cora에서 MLP를 훈련시켜 본다 !

⑴ MLP 모델을 만들고 올바른 레이어가 있는지 확인하기 위해서 출력하기



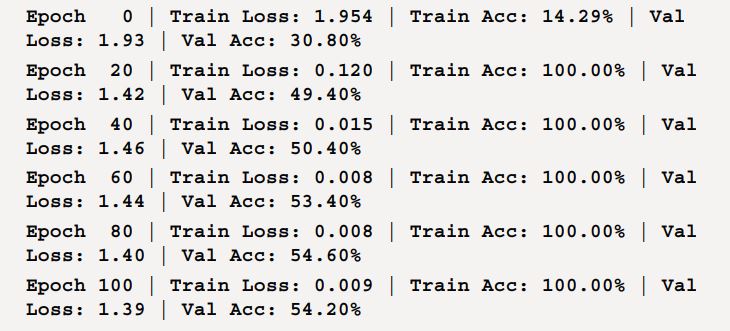
⑵ 다음은 출력 결과임



⑶ 올바른 특성 수를 얻은 것을 확인하였음. 이 모델을 100 에포크 동안 훈련함.



⑷ 훈련 루프에서 인쇄되는 메트릭은 다음과 같음



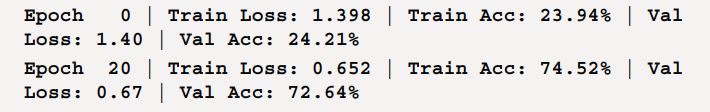
⑸ 마지막으로 정확도를 기준으로 모델의 성능을 평가함

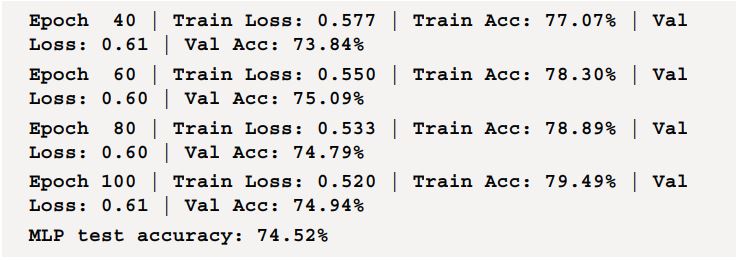


⑹ 테스트 데이터에서 다음과 같은 정확도 점수를 얻음



⑺ 이제 Facebook Page-Page 데이터셋에 대해 동일한 프로세스를 반복하고 출력 결과를 확인함





→ 이러한 데이터셋은 몇 가지 측면에서 유사하지만, 얻는 정확도 점수가 크게 다르다는 것을 볼수 있음. 이러한 차이를 네트워크 토폴로지와 노드 특성을 동일한 모델에서 결합할 때 흥미로운 비교 대상으로 삼을 것임 !!

**3) 바닐라 그래프 신경망을 사용한 노드 분류**

- 잘 알려진 GNN 아키텍처를 직접 소개하는 대신, GNN에 대한 사고 과정을 이해하기 위해 자체 모델을 만들어 보자.

- 먼저, 간단한 선형 레이어의 정의로 돌아가야 함.

- 기본적인 신경망 레이어는 입력 벡터 에 대한 선형변환 = 에 해당

- 여기서 는 노드 A의 입력 벡터, W는 가중치 행렬

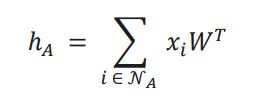
- PyTorch에서는 이 방정식을 torch.mm() 함수로 구현하거나 다른 매개변수(바이어스 등)를 추가하는 nn.Linear 클래스로 구현 가능 !

- 우리의 그래프 데이터셋에서 입력 벡터는 노드 특성임.

- 이것은 노드가 완전히 분리된 것을 의미

- 노드를 이해하려면 해당 노드의 이웃을 살펴봐야 함 !

- 노드의 이웃집합을 정의해보자. 그래프 선형 레이어는 다음과 같이 작성할 수 있음 :



- 이 식의 여러가지 변형이 가능함. 예를 들어, 중앙 노드에 대한 가중치 행렬 과 이웃에 대한 다른 가중치 행렬 를 가질 수 있음.

- 노드마다 이웃 수가 달라질 수 있으므로 이웃마다 가중치 행렬을 가질 수는 X

- 신경망에 대해 이야기하고 있으므로 이전의 방정식을 각 노드에 적용할 수는 X

- 대신 훨씬 효율적인 행렬 곱셈을 수행 !

- 예를 들어, 선형 레이어의 방정식은 입력행렬 X에 대한 행렬곱셈으로 다시 쓸 수 O

- 우리의 경우에는 인접행렬 A가 그래프 내 모든 노드 간의 연결을 포함하고 O

- 입력 행렬을 이 인접행렬로 곱하면 이웃 노드의 특성이 직접 합산됨 !

- 중앙 노드도 이 작업에 포함되도록 인접 행렬에 self loop를 추가할 수 O

- 이 갱신된 인접행렬을 Ã = A + 라 부르도록 함 !

- 우리의 그래프 선형 레이어는 다음과 같이 다시 작성할 수 O



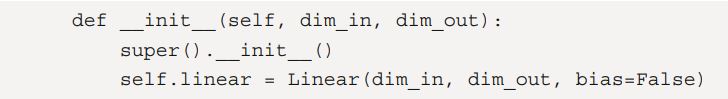
☞ 이전에 구현한 레이어를 PyTorch Geometric에서 구현하여 테스트 해보자

- 이후 이를 일반적인 레이어처럼 사용하여 GNN을 구축할 수 있을 것!

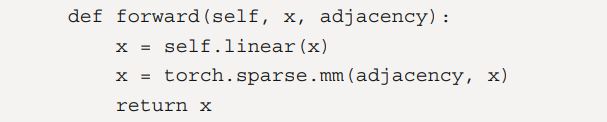
⑴ 먼저, torch.nn.Module의 하위 클래스인 새 클래스를 만들기



⑵ 이 클래스는 입력 및 출력의 특성 수에 대한 두 매개변수 dim\_in 및 dim\_out을 가져와서 편향없는 기본 선형 변환을 추가함



⑶ 우리는 두 가지 작업을 수행함. 선형변환과 인접행렬 Ã와의 곱셈

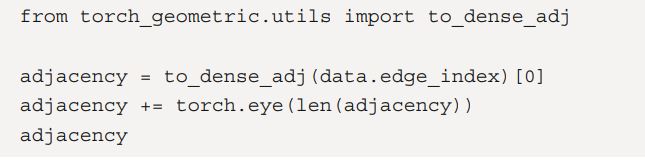


- 바닐라 GNN을 생성하기 전에, 데이터셋에서 (data.edge\_index) edge index를 좌표 형식에서 밀도가 높은 인접 행렬로 변환해야 함 !

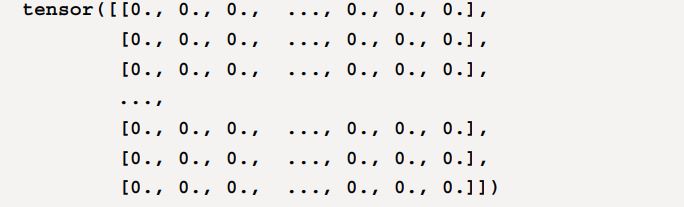
- 또한 self loop를 포함해야 함 !

- 그렇지 않으면 중심 노드가 자신의 임베딩에 고려되지 X

⑷ 이는 to\_den\_adj()와 torch.eye() 함수를 사용하여 쉽게 구현할 수 O

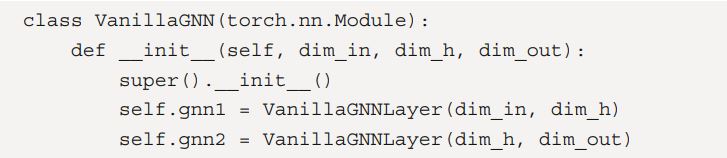


- 이제 인접 행렬의 모습을 살펴보자

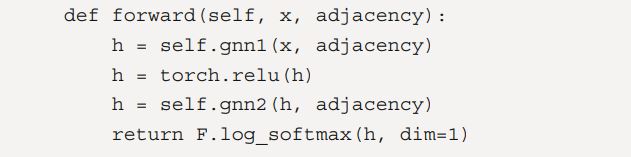


- 이 텐서에서는 희소 행렬이기 때문에 모두 0만 보임. 보다 자세한 출력은 몇 개의 연결된 노드를 보여줌 (1료 표시). 이제 전용 레이어와 인접행렬이 준비되었으므로, 바닐라 GNN의 구현은 MLP의 구현과 매우 유사 !

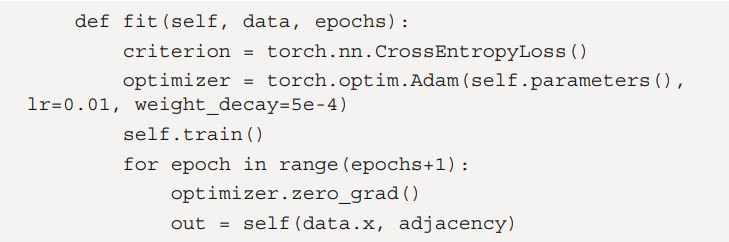
⑸ 두 개의 바닐라 그래프 선형 레이어를 가진 새 클래스를 만들기

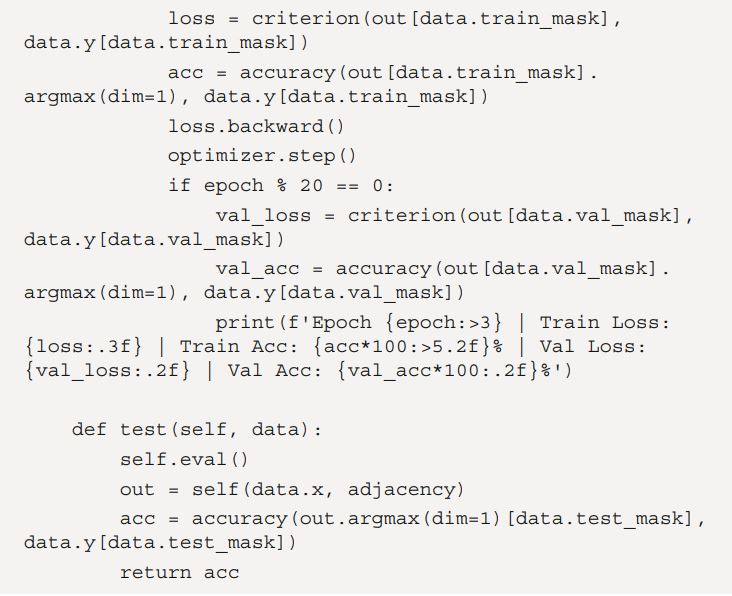


⑹ 우리는 새로운 레이어로 동일한 작업을 수행함. 이 레이어는 이전에 계산한 인접 행렬을 추가 입력으로 받음

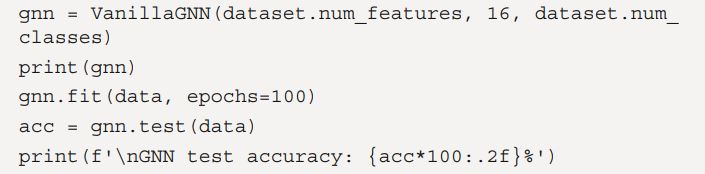


⑺ fit() 및 test() 메서드는 정확히 동일한 방식으로 작동함

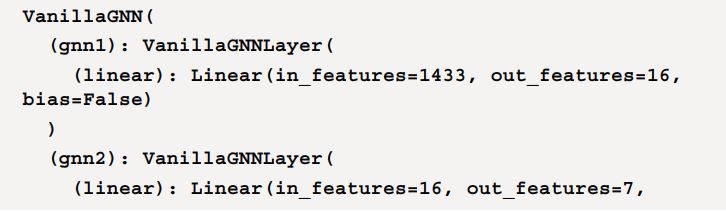


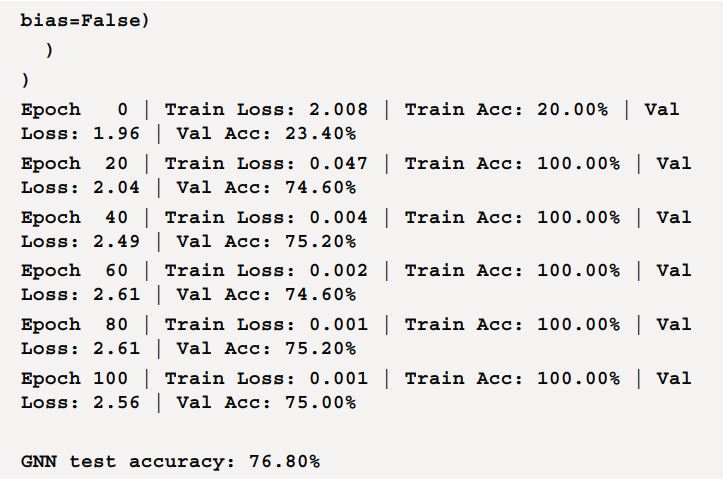


⑻ 다음과 같이 모델을 생성, 학습 및 평가할 수 있음



⑼ 이렇게 하면 다음 출력이 생성됨

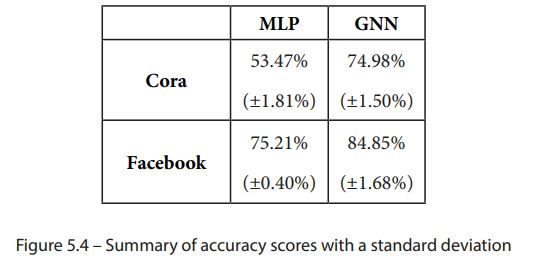




☞ Facebook Page-Page 데이터셋에서도 동일한 훈련 과정을 복제함.

- 비교 가능한 결과를 얻기 위해 각 데이터 셋의 각 모델에 대해 실험을 100번 반복

- 아래 표는 결과를 요약한 것 !



∴ MLP가 Cora에서 성능이 좋지 않음을 알 수 O

→ Facebook Page-Page 데이터셋에서는 조금 더 나은 성능을 보이지만 여전히 바닐라 GNN에 뒤처짐 !

- 이러한 결과는 노드 특징에 위상 정보를 포함하는 것의 중요성을 보여줌

→ 우리의 GNN은 타당한 데이터셋 대신 각 노드의 전체 이웃을 고려하므로, 이러한 예제에서 정확도가 10-20% 향상됨

→ 이 아키텍처는 아직 초기 단계이지만, 우리에게 그것을 개선하고 더 나은 모델을 구축하는 지침을 제공

요약

- 이 장에서는 바닐라 신경망과 GNN(그래프 신경망) 사이의 결핍된 연결고리에 대해 배움.

- 직관과 약간의 선형대수를 사용하여 자체 GNN 아키텍처를 구축함.

- 과학 문헌에서 사용되는 두 가지 인기있는 그래프 데이터셋을 탐색하여 두 아키텍처를 비교함.

- 마지막으로 PyTorch에서 이를 구현하고 그 성능을 평가함.

- 결과 : 직관적인 GNN 버전도 두 데이터셋 모두에서 MLP를 완전히 능가함 !

☞ 다음 6장에서는 그래프 합성곱 신경망을 사용하여 바닐라 GNN 아키텍처를 올바르게 정규화 함.