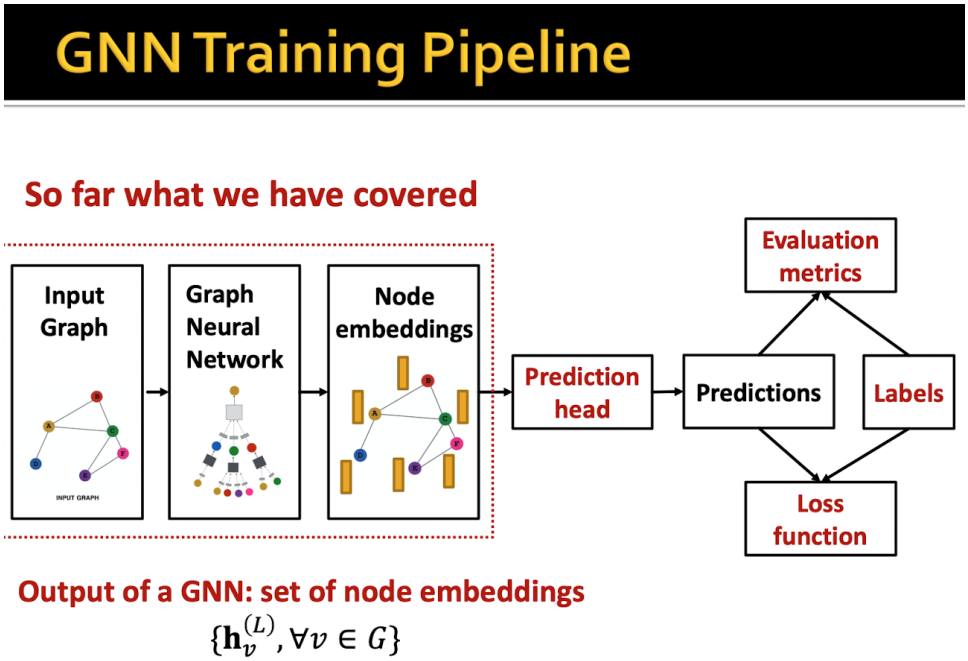
8주차 강의노트

그래프 기본 요소들 정리

▼ 2.Prediction with GNNs



- Intro] 이제 그래프가 학습가능한 상태가 되었다면, 우리는 어떻게 GNN을 학습시킬 것인가?

  학습은 예측한 값과 기존 레이블의 오차를 줄여가는 방향으로 진행된다. 이를 위해 우선 GNN을 거친 Node embedding 된 값들을 이용해서 Prediction head를 구할 것이다.

  Idea는 다른 task levels에는 다른 prediction heads 이 필요하다는 것으로 Node, Edge , Graph 레벨에서 각각 prediction head를 구하는 것을 살펴볼 것

이다.

- Prediction Heads : Node-level

  GNN 의 아웃풋인 node embedding 값 h_v^L 은 d-dim 이다. 그러나 우리가 k-way prediction을 한다고 가정하면, k-dim의 prediction head가 필요하므로 h_v^L에 k*d dim의 W^H 를 곱해서 k-dim의 $yhat_v$으로 매핑해준다.

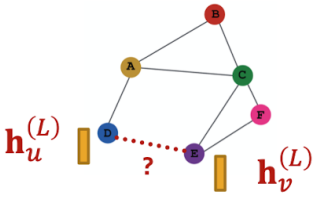
  그러면 이제 k-dim 의 레이블과 loss를 계산해줄 수 있게 된다.

  - $\hat{\boldsymbol{y}}_{\boldsymbol{v}} = \text{Head}_{\text{node}}(\mathbf{h}_v^{(L)}) = \mathbf{W}^{(H)}\mathbf{h}_v^{(L)}$
    - $\mathbf{W}^{(H)} \in \mathbb{R}^{k*d}$: We map node embeddings from $\mathbf{h}_v^{(L)} \in \mathbb{R}^d$ to $\hat{\boldsymbol{y}}_v \in \mathbb{R}^k$ so that we can compute the loss

- Prediction Heads : Edge-level

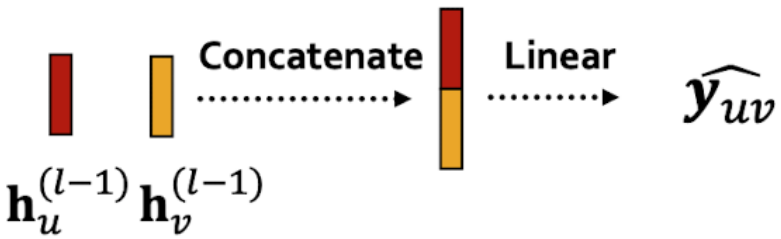
  아래 그림과 같이 한쌍의 노드 임베딩을 사용해 prediction 을 만든다.

$$\hat{\boldsymbol{y}}_{\boldsymbol{uv}} = \text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$$



$\mathbf{h}_u^{(L)}$ $\mathbf{h}_v^{(L)}$

1. Concatenation + Linear



$\mathbf{h}_u^{(l-1)}$ $\mathbf{h}_v^{(l-1)}$ $\hat{\boldsymbol{y}}_{uv}$

l-1 레이어에서의 u와 v 노드의 임베딩 값을 concat 한 후, 해당 값을 Linear 함수를 통해 2d-dim → k-dim embeddings 로 매핑 시킨다.

2. Dot product

$$\hat{y}_{uv} = (\mathbf{h}_u^{(L)})^T \mathbf{h}_v^{(L)}$$

이 방법은 1-way-prediction 에서만 작동한다. (eg. 두 노드 사이 edge 존재 여부 예측)

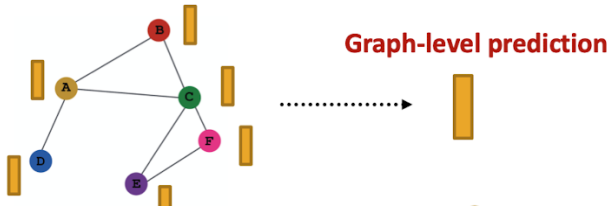
- Similar to multi-head attention: $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(k)}$ trainable

$$\hat{y}_{uv}^{(1)} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(1)} \mathbf{h}_v^{(L)}$$
$$\ldots$$
$$\hat{y}_{uv}^{(k)} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(k)} \mathbf{h}_v^{(L)}$$
$$\hat{y}_{uv} = \text{Concat}(\hat{y}_{uv}^{(1)}, \ldots, \hat{y}_{uv}^{(k)}) \in \mathbb{R}^k$$

그러나 mullti-head attention과 비슷한 방법으로 병렬적으로 k개의 웨이트 값을 각각 곱해서 값을 구한 후 concat 해주면 k-dim의 prediction head를 구할 수 있다. 이 때 W(웨이트)값들은 학습 가능하기 때문에 축소, 확대 , 회전 , 변환이 가능하다.

- Prediction Heads : Graph-level

$$\hat{y}_G = \text{Head}_{\text{graph}}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$



그래프의 모든 노드 임베딩을 사용하여 예측값을 구할 수 있다. 이 때 앞서 배운 GNN에서 aggregation 하는 과정과 유사하다.

아래와 같이 3가지 global pooling 방법들이 있으나 , 이 방법은 작은 그래프에서만 잘 작동한다. 그렇다면 큰 그래프에서 더 잘 작동할 수 있는 방법은 없을까?

- Options for $\text{Head}_{\text{graph}}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$
- **(1) Global mean pooling**

$$\hat{\boldsymbol{y}}_G = \text{Mean}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

- **(2) Global max pooling**

$$\hat{\boldsymbol{y}}_G = \text{Max}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

- **(3) Global sum pooling**

$$\hat{\boldsymbol{y}}_G = \text{Sum}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

→ Global Pooling의 문제점 : 분명 다른 그래프 구조임에도 구분하지 못하는 일이 발생 ,

e.g) node embedding forG1 = {-1,-2,0,1,2} , node embedding for G2 = {-10,-20,0,10,20}

- Prediction for G1:y_hat of G = Sum({-1,-2,0,1,2}) = 0
- Prediction for G2:y_hat of G = Sum({-10,-20,0,10,20}) = 0
- 즉 , 엄연히 다른 두 가지 그래프를 구분하지 못하는 일이 일어난다!

그래서 Hierarchical Global Pooling(계층적 풀링)의 개념이 등장한다.

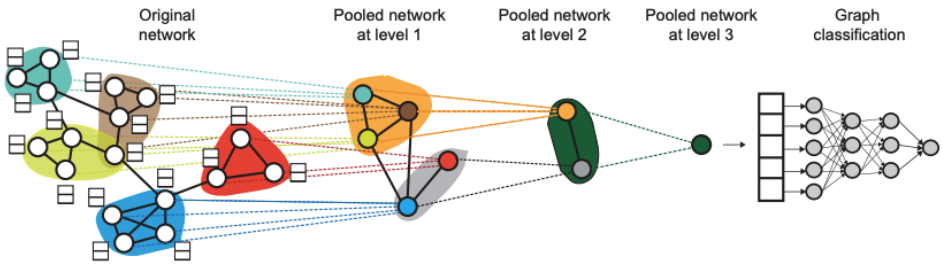
우리는 ReLU(Sum(-)) 을 통해 aggregate 할것이다. 예를 살펴보면,

e.g) node embedding for G1 = {-1,-2,0,1,2} , node embedding forG2 = {-10,-20,0,10,20}

- Round1 : y_hat_a = ReLU(Sum({-1,-2})) = 0 , y_hat_b = ReLU(Sum({0,1,2})) = 3
- Round2 : y_hat of G = ReLU(Sum({y_a , y_b})) = 3
- 그리고 G2에 대해서 다시 해주면, 두 개의 값이 달라 두 그래프가 구분되는 것을 볼 수 있다.
- Round1 : y_hat_a = ReLU(Sum({-10,-20})) = 0 , y_hat_b = ReLU(Sum({0,10,20})) = 30
- Round2 : y_hat of G = ReLU(Sum({y_a , y_b})) = 30

- Hierachical Pooling
  - DiffPool idea : Hierarchically pool node embedding

    However, current GNN methods are inherently flat and do not learn hierarchical
    representations of graphs—a limitation that is especially problematic for the task
    of graph classification, where the goal is to predict the label associated with an
    entire graph. (전체 그래프에 대해 각 레이어 별로 군집에 대한 레이블이 할당된다. 즉 그래프 분류 문제에서 지도학습이 가능해진다.)



    또한 각 노드에 대한 임베딩(GNN A)와 노드가 속한 클러스터에 대한 계산(GNN B)가 독립적인 GNN으로 이뤄지며 각 단계마다 병렬적으로 계산할 수 있다.

    각 풀링 레이어에선

    - GNN A 를 통해 임베딩 된 노드 임베딩 값들을 GNN B 를 이용해 묶어준다.
    - 각 클러스터에 대한 싱글 노드를 만들어준 후, 새로운 네트워크를 만들기 위해 엣지를 연결해서 클러스터링 해준다.

▼ 3. Train GNN

이제 예측한 값들과 레이블들을 비교해서 Loss 를 계산해주고 , 평가지표를 만드는 것일 공부할 것이다.

- Supervised learning on graphs

  그래프에서 지도학습은 외부의 정보가 레이블로 온다.(e.g 약물 유사성 예측, 분자 독성 유무)

- Unsupervised learning on graphs

그래프 자체 정보로 'self-supervised'한다. (e.g 연결 예측)

- 때론 둘 사이 차이가 모호하기도 하다.

  우린 여전히 비지도학습에서 '지도'할 수 있다. (e.g 노드 클러스터링 상관관계 예측을 위한 GNN 학습)

  그래서 우린 비지도 학습 대신 자체학습이라고 대체해서 부를 수 있다.

지도학습 레이블은 특수한 케이스에서 온다.

- node labels y_v : 인용네트워크에서 노드가 속한 주제가 어딘 지
- edge labels y_uv : 거래 네트워크에서 엣지가 사기인지 아닌 지
- graph labels y_G : 분자 구조에서 약물 유사도

조언하나 하자면 , 이것이 쉽게 작동하기 위해 노드, 엣지, 그래프 레이블의 작업을 줄여야한다.

eg) 클러스터에 속한 노드에 대해 , 우리는 클러스터를 레이블로 취급할 수 있다.

비지도 학습에 대해선 외부 레이블이 없다는 문제가 있지만, 이것에 대해선 self-supervise를 통해 그래프 내부에서 지도 시그널을 찾을 수 있다. 예를 들어 GNN은 아래 것들을 예측할 수 있다.

- node-level : 노드에 대한 통계량 : 클러스터링 계수 나 페이지 랭크
- edge-level : 두 노드 간 연결 예측
- graph-level : 그래프 사이 동형인 지 아닌 지 예측

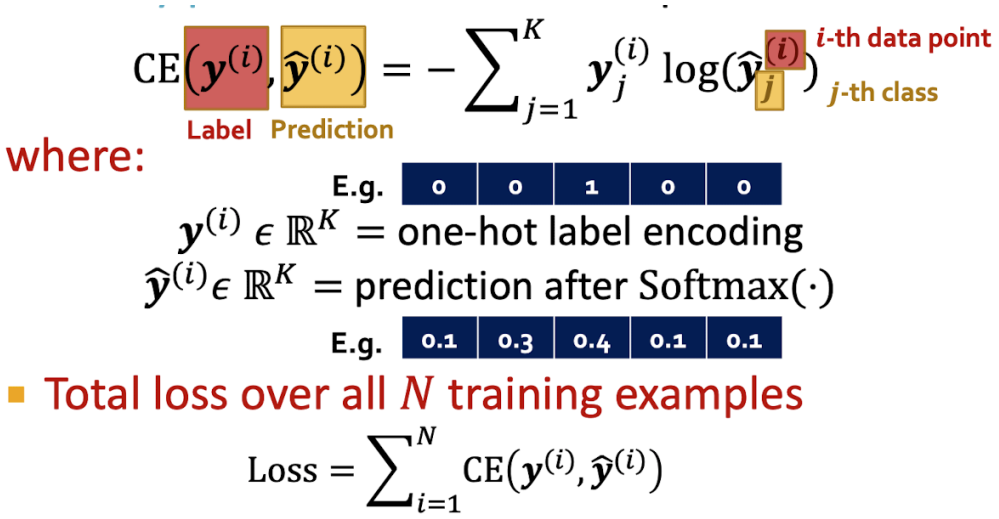
위의 경우들은 외부 레이블이 필요 없다.

- How do we compute the final loss?

  우리가 N 개의 데이터 포인트를 가지고 있다고 하자. 각 데이터 포인트에 대해 노드 ,엣지 , 그래프 레벨의 예측 값들을 구할 수 있다.

  - 분류문제에선 데이터 포인트는 이산값으로 나타나고, 회귀 문제에선 연속된 값으로 나타난다. 다만 이 둘 사이에서 loss function 과 evaluation metrics의 차이가 있다.
  - 분류 문제에서 loss

6과에서 배웠듯이 cross entropy(CE) 를 일반적으로 사용한다. 계산은 k-way 라고 할 때, k개의 클래스에 대해 CE를 계산해준다. 각각의 데이터 포인트에 대해 계산한 CE 들을 모두 더해주면 Loss를 산출할 수 있다. 이때, 레이블의 값으로 원핫 인코딩을 사용함으로 1인 원소 외에는 다 0으로 값이 나와 계산양이 많지는 않다. 또한 완벽한 예측으로 예측값이 1이 나오면 CE는 0이 된다.



- 회귀문제에서 loss

  L2 loss라고 불리는 MSE(mean square error)를 사용한다. k-way 에 대해 각 클래스에서의 오차의 제곱을 더해서 MSE를 구해준 후, 각 데이터 포인트마다 구해준 MSE를 더해주면 Loss를 산출한다.

  이 때, MSE를 사용하는 이유는 smooth, continue하므로 derivative를 얻기 쉽고 항상 양수값이기 때문이다.

- How do we measure the success of a GNN?

  accuracy 와 ROC AUC 두가지가 있다.

  - Evaluation Metrics : Regression

    GNN을 위한 표준 평가 매트릭스 사용 : RMSE , MAE

  - Evaluation Metrics : Classification
    - Multi-class classification

$$\frac{1\left[\text{argmax}\left(\hat{\boldsymbol{y}}^{(i)}\right) = \boldsymbol{y}^{(i)}\right]}{N}$$

(argmax는 최댓값의 인덱스) 위 식과 같이 간단하게 정확도를 나타낸다. 그러나 accuracy는 imbalance한 데이터에서 좋은 지표가 되지 못한다. 왜냐면 긍정 99개와 부정 1개가 있을 때 다 긍정이라고 말하는 이 허접한 로직이 99퍼의 정확도를 띄기 때문이다.

- Binary classification

  Accuracy , Precision , Recall 그리고 스레스홀드에 구애받지 않는 ROC 커브와 AUC

Confusion matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

eg) 암 환자와 정상인 사람 구분 하는 문제 ,

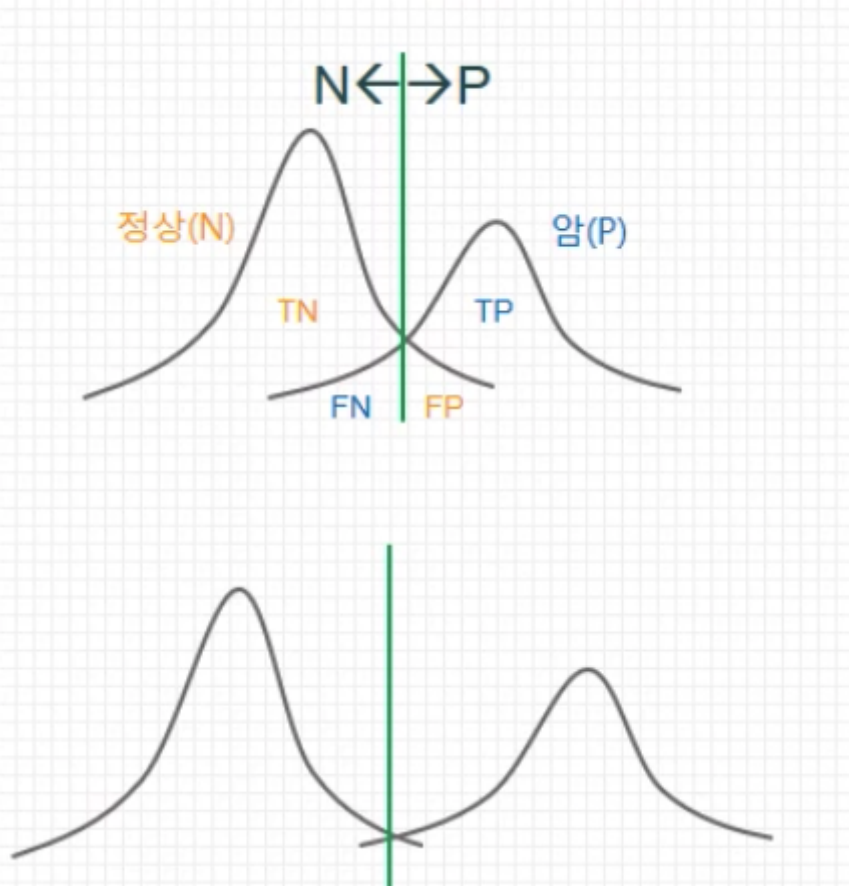
Precision : 암환자라고 예측 했을 때 실제 암환자 일 확률 , 예를 들어 100명 예측했는데 1명이 정상이었으면 99%

Recall : 실제 암환자 중 암환자라고 예측할 확률 , 예를 들어 100명의 암환자가 있는데 98명을 예측하고 두명은 일반인이라고 판단했을 경우 98%

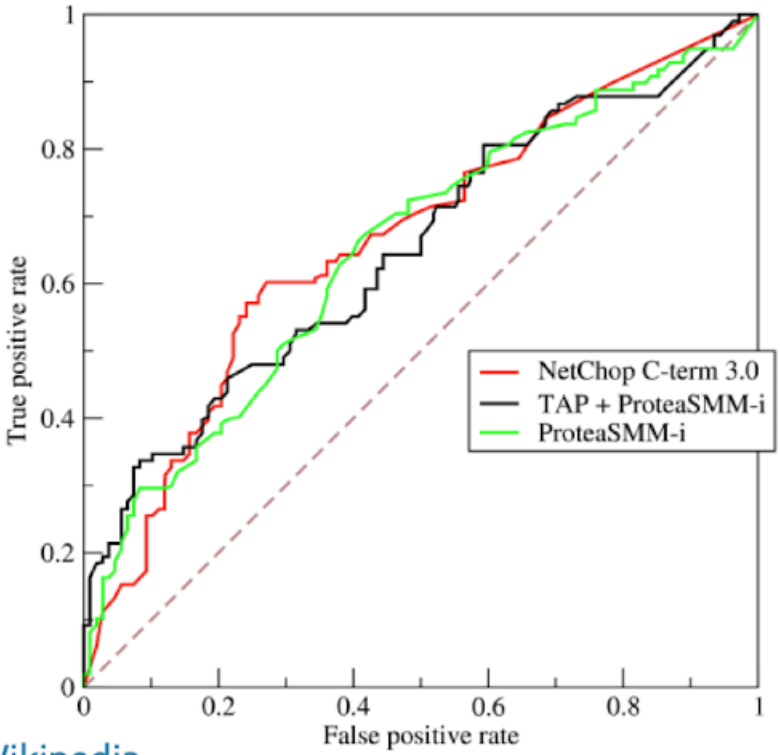
F1-Score : 위의 두 지표를 적절히 조합한 값.

아래 그래프를 보면 초록색 직선이 판단의 기준이라고 했을 때, 오른쪽은 내가 암환자라고 판단 , 왼쪽은 정상인이라고 판단한 것 , 만약 아래의 그래프와 같이 초록색선 부근에 걸쳐있는 영역이 많이 없다면 헷갈리는 케이스가 많이 없는 영역(즉 오판할 확률 낮다.) 그러나 위와 같은 그래프라면 극단적으로 초록색 결정선을 맨 왼쪽으로 옮기면 암환자는 모두 맞출수있지만 암환자가 아닌데 오진되는 경우 발생하고, 오른쪽 끝으로 옮기면 정상인은 모두 고르지만 암환자인데 정상인 판정 받는다. 위 두가지 오류 케이스를 모두 줄인 것이 ROC 커브 그래프의 왼쪽 상단이다. 그래서 이 그래프 볼때는 왼쪽 상단에 가까울 수록 좋은 지표이다. 이 때 정량적으로 이 그래프의 성능을 판단하는 지표가 AUC인데 ROC 커브 아래 영역의 면적의 값을 나타낸다. 이 값이 클수록 더 좋은 성능을 내는 것이다.

이 그래프에서 가운데 직선 값은, 암환자를 정확히 예측하는 확률도 올라가면서 동시에 정상인을 암으로 판정하는 횟수도 오라가는데, 이러한 경우는 좋지 못한 성능을 낸다고 볼 수 있다.



TPR



e Credit: Wikipedia

FPR