Spotify Playlist Recommendation AI

Team GNNie

Contents

Our Teams

이하윤
Lee Hayun

BOAZ 분석 22기

이혜승
Lee Hyeseung

BOAZ 분석 22기

장서연
Jang Seoyeon

BOAZ 분석 22기

허지윤
Heo Jiyun

BOAZ 분석 22기

Introduction

About Project

특정 음악 선호도에 맞는 플레이리스트를 자동으로 생성하는 방법을 제안하는 프로젝트

⇒ 좋아하는 노래를 포함할 뿐만 아니라 **사용자의 취향에 맞는 새로운 트랙**을 소개하는 것이 목표

기존의 추천시스템

- User-based Filtering
- Item-based Filtering
- etc..

⇒ 낮은 예측 성능 + **Cold Start**등의 문제가 발생

**Cold Start: 사용자의 상품 구매 또는 평가 이력과 같은 기존 행동 정보가 없는 경우, 예측이 어려운 상태*

기존의 추천 시스템의 한계를 해결하는 **GNN** 모델로 보다 효과적인 플레이리스트 추천 시스템을 구축하자!

Dataset

Spotify Million Playlist Dataset

2010년 1월부터 2017년 10월 사이에 생성된 트랙 정보, 아티스트 세부 정보 및 앨범 데이터가 포함된 100만 개의 Spotify 재생 목록으로 구성

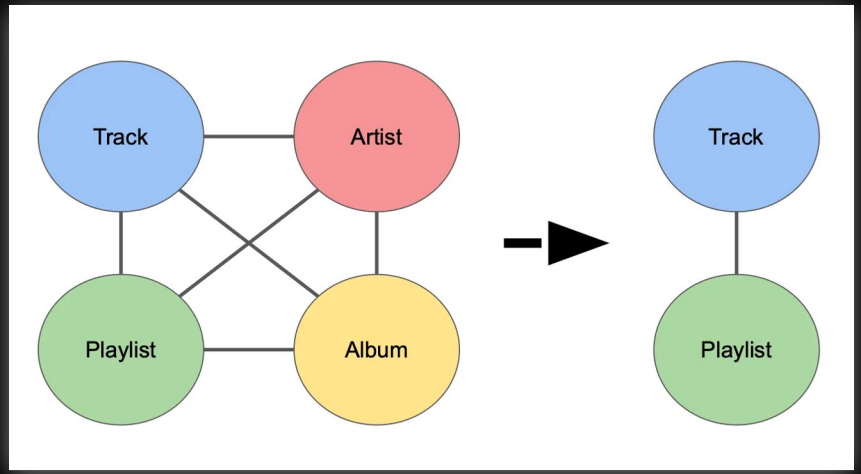
'세계에서 가장 큰 음악 Playlist 데이터 세트'


```
{'name': 'Throwbacks',
 'collaborative': 'false',
 'pid': 0,
 'modified_at': 1493424000,
 'num_tracks': 52,
 'num_albums': 47,
 'num_followers': 1,
 'tracks': [{'pos': 0,
   'artist_name': 'Missy Elliott',
   'track_uri': 'spotify:track:0UaMYEvWZi0ZqiDOoHU3YI',
   'artist_uri': 'spotify:artist:2wIVse2owClT7go1WT98tk',
   'track_name': 'Lose Control (feat. Ciara & Fat Man Scoop)',
   'album_uri': 'spotify:album:6vV5UrXcfyQD1wu4Qo2I9K',
   'duration_ms': 226863,
   'album_name': 'The Cookbook'},
  {'pos': 1,
   'artist_name': 'Britney Spears',
   'track_uri': 'spotify:track:6I9VzXrHxO9rA9A5euc8Ak',
   'artist_uri': 'spotify:artist:26dSoYclwsYLMAKD3tpOr4',
   'track_name': 'Toxic',
   'album_uri': 'spotify:album:0z7pVBGOD7HCIB7S8eLkLI',
   'duration_ms': 198800,
   'album_name': 'In The Zone'},
```

Models

그래프 신경망(Graph Neural Networks, GNNs), 왜 사용할까?

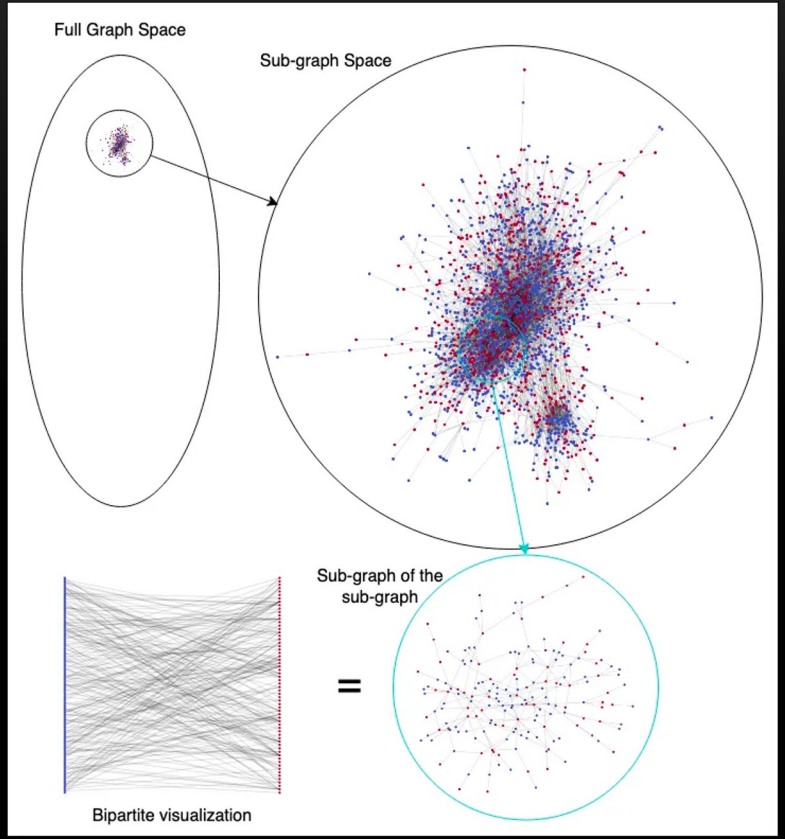
음악 플레이리스트와 같이 **크고 상호 연결**된 데이터셋에서 작동하는 **추천 시스템**에 이상적인 선택!



Models

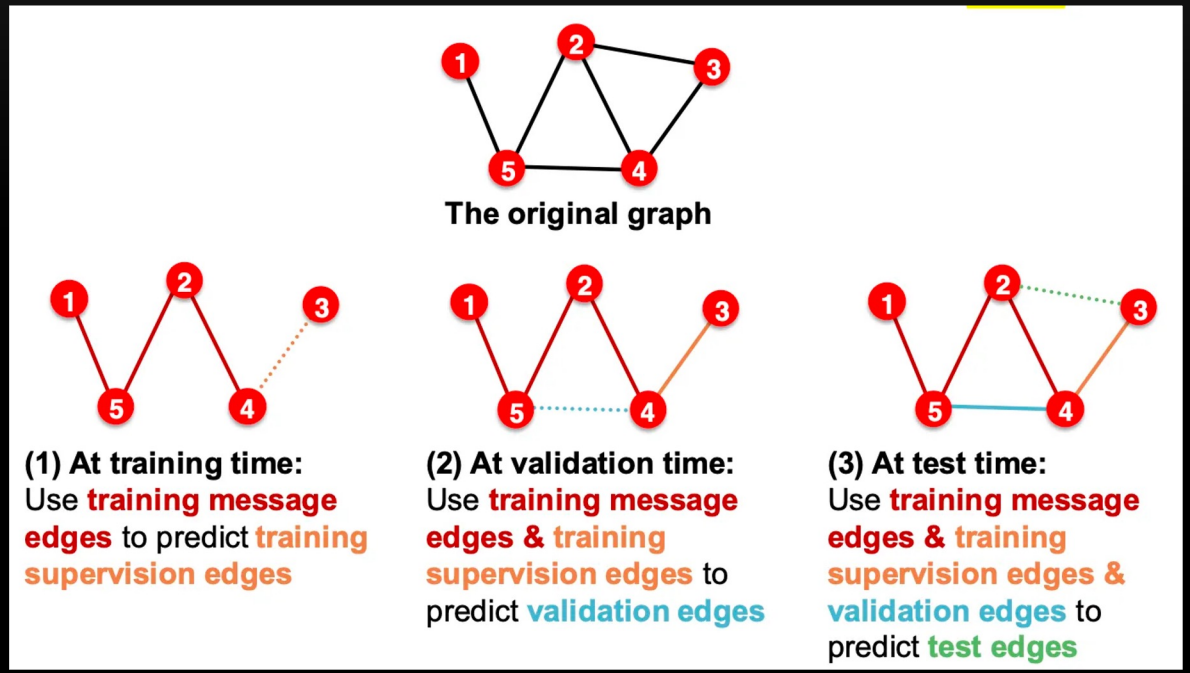
Visualizing Graph

K-Core :
하위 그래프의 모든 정점이 **최소한 K의 차수**를 갖는 그래프의 **하위 그래프**로,
가장 많이 연결되고 긴밀하게 통합된 정점이 위치한
네트워크의 핵심을 나타낸다.



Models

Train-Test Split

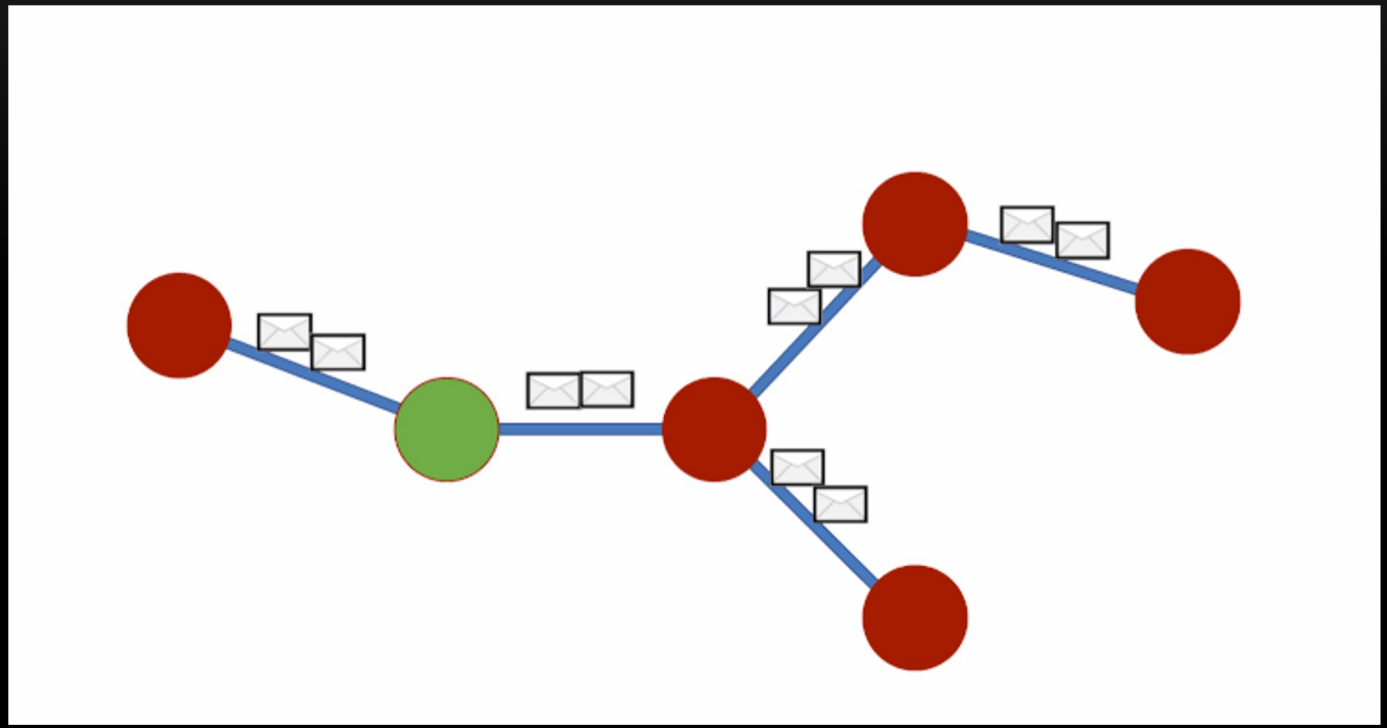


Models

그래프 신경망(Graph Neural Networks, GNNs), 왜 사용할까?

대부분의 기계 학습 방법이
순차적 데이터(예: 텍스트)나
격자 데이터(예: 이미지)에
중점을 두고 있으며,
이전 신경망 모델들은
그래프 데이터의 복잡한 구조를
다루는 데 어려움을 겪는다는 점

반면, GNN은
데이터의 그래프 구조를 활용하여
노드와 엣지 간의
고차 의존성뿐만 아니라
노드 및 엣지의 특성을 포착하여
노드, 엣지 또는 (하위)그래프의
임베딩이나 표현을 학습할 수 있다!

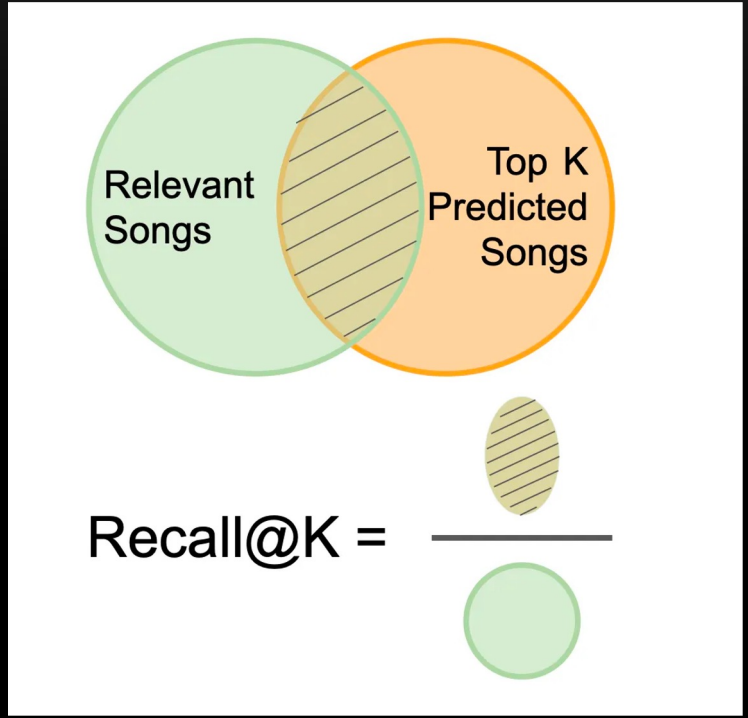


Models

Defining our loss

Recall@K

상위 K개의 추천 중
관련 있는 트랙의 비율
(이 경우, 플레이리스트에
포함된 트랙)



Models

LightGCN (LGConv)

Graph Attention network (GATConv)

GraphSAGE (SAGEConv)

Code_Designing our model

```
class GCN(torch.nn.Module):
    def __init__(
        self,
        num_nodes: int,
        embedding_dim: int,
        num_layers: int,
        alpha: Optional[Union[float, Tensor]] = None,
        alpha_learnable = False,
        conv_layer = "LGC",
        name = None,
        **kwargs,
    ):
        super().__init__()
        alpha_string = "alpha" if alpha_learnable else ""
        self.name = f"LGCN_{conv_layer}_{num_layers}_e{embedding_dim}_nodes{num_nodes}_{alpha_string}"
        self.num_nodes = num_nodes
        self.embedding_dim = embedding_dim
        self.num_layers = num_layers

        if alpha_learnable == True:
          alpha_vals = torch.rand(num_layers+1)
          alpha = nn.Parameter(alpha_vals/torch.sum(alpha_vals))
          print(f"Alpha learnable, initialized to: {alpha.softmax(dim=-1)}")
        else:
          if alpha is None:
              alpha = 1. / (num_layers + 1)

          if isinstance(alpha, Tensor):
              assert alpha.size(0) == num_layers + 1
          else:
              alpha = torch.tensor([alpha] * (num_layers + 1))

        self.register_buffer('alpha', alpha)

        self.embedding = Embedding(num_nodes, embedding_dim)

        # initialize convolutional layers
        self.conv_layer = conv_layer
        if conv_layer == "LGC":
          self.convs = ModuleList([LGConv(**kwargs) for _ in range(num_layers)])
        elif conv_layer == "GAT":
          # initialize Graph Attention layer with multiple heads
          # initialize linear layers to aggregate heads
          n_heads = 5
          self.convs = ModuleList(
              [GATConv(in_channels = embedding_dim, out_channels = embedding_dim, heads = n
```

1) GCN 모델

def __init__

- GNN의 구조를 정의하고 초기 파라미터 설정
- 주요 구성 요소
  - `num_nodes` : 그래프의 노드 수
  - `embedding_dim` : 노드의 임베딩 차원
  - `num_layers` : GCN 레이어의 수
  - `alpha` : GCN 레이어 간의 가중치를 제어하는 매개변수로, 학습 가능한지 여부에 따라 결정된다
  - `alpha_learnable` : `True` 이면 `alpha` 값을 학습 가능하게 만든다
  - `conv_layer` : 사용할 GCN 레이어의 종류를 나타냅니다. "LGC" (LightGCN), "GAT" (Graph Attention), "SAGE" (GraphSAGE) 중 하나를 선택할 수 있다
  - `name` : 모델의 이름을 나타냅니다.
  - `kwargs` : GCN 레이어 및 기타 구성 요소에 대한 추가적인 매개변수입니다.
- 주요 동작
  - `alpha_learnable`
    - `True` 인 경우, `alpha` 값을 무작위로 초기화하고 학습 가능한 파라미터로 설정한다
    - 그렇지 않으면, `alpha` 값을 초기화하고 학습 불가능한 상수로 설정
  - GCN 레이어의 종류에 따라 해당 레이어들을 초기화
    - "LGC"의 경우 `LGConv` , "GAT"의 경우 `GATConv` 및 `Linear` , "SAGE"의 경우 `SAGEConv` 를 사용
  - 나머지 부분은 각종 설정 및 초기화 작업을 수행.

Code_Designing our model

```
def reset_parameters(self):
    torch.nn.init.xavier_uniform_(self.embedding.weight)
    for conv in self.convs:
        conv.reset_parameters()
```

def reset_parameters(self)

- GCN 모델의 가중치를 초기화하는 `reset_parameters` 메서드를 정의
- Xavier 초기화 방법 사용
  - Xavier 초기화는 가중치를 적절한 분포로 초기화하여 학습을 안정적으로 진행시키는데 사용
- 주요 동작
  - 임베딩 레이어의 가중치를 초기화: `self.embedding.weight` 의 가중치를 Xavier 초기화를 사용하여 초기화
  - 각 GCN 레이어의 가중치를 초기화: 각 레이어의 `reset_parameters` 메서드를 호출

```
def get_embedding(self, edge_index: Adj) -> Tensor:
    x = self.embedding.weight

    weights = self.alpha.softmax(dim=-1)
    out = x * weights[0]

    for i in range(self.num_layers):
        x = self.convs[i](x, edge_index)
        if self.conv_layer == "GAT":
          x = self.linears[i](x)
        out = out + x * weights[i + 1]

    return out
```

def get_embedding(self, edge_index: Adj)

- 노드의 임베딩을 얻는 `get_embedding` 메서드를 정의
- 주요 동작 단계
  1. `self.embedding.weight` 에서 초기 노드 임베딩인 `x` 를 가져온다
  2. `self.alpha` 에 대한 softmax를 계산하여 각 GCN 레이어의 가중치를 얻는다
  3. 초기 노드 임베딩에 첫 번째 가중치를 곱하여 초기값 `out` 을 설정
  4. 각 GCN 레이어에 대해 다음을 반복:
     - 현재 레이어의 GCN 연산을 수행하여 `x` 를 갱신합니다.
       - `self.convs[i]` 는 `i` 번째 GCN 레이어
       - `x` 와 엣지 인덱스 `edge_index` 를 받아서 GCN 연산을 수행하고 결과를 반환
     - GCN 레이어가 "GAT"인 경우, 해당 레이어의 결과에 대해 추가적으로 선형 변환을 수행합니다.
     - 이전 레이어의 결과인 `out` 에 현재 레이어의 결과를 더하고, 해당 레이어의 가중치를 곱하여 갱신합니다.
  5. 최종적으로 갱신된 `out` 이 최종 노드 임베딩으로 반환됩니다.

```
def initialize_embeddings(self, data):
  # initialize with the data node features
    self.embedding.weight.data.copy_(data.node_feature)
```

def initialize_embeddings(self, data):

- GCN 모델의 초기 임베딩을 데이터의 노드 특성으로 설정
- 이 메서드는 모델이 학습을 시작하기 전에 노드의 초기 임베딩을 데이터의 실제 특성값으로 초기화하는 데 사용
- 주요 구성
  - `self.embedding.weight.data` 는 GCN 모델의 임베딩 가중치를 나타낸다. 이 가중치는 각 노드에 대한 임베딩을 저장함
  - `data.node_feature` 는 입력 데이터 `data` 에서 노드의 실제 특성값을 나타냄

Code_Designing our model

```python
def forward(self, edge_index: Adj,
            edge_label_index: OptTensor = None) -> Tensor:
    if edge_label_index is None:
        if isinstance(edge_index, SparseTensor):
            edge_label_index = torch.stack(edge_index.coo()[:2], dim=0)
        else:
            edge_label_index = edge_index

    out = self.get_embedding(edge_index)

    return self.predict_link_embedding(out, edge_label_index)
```

def forward()

- GCN 모델의 순전파(`forward`) 메서드; 모델에 입력 데이터를 전달하여 예측을 수행하는 역할
- 주요 구성
  - `edge_index` : 그래프의 엣지 인덱스로, 각 엣지에 대한 정보를 나타냄
  - `edge_label_index` : 옵션으로, 모델이 예측할 엣지의 인덱스를 나타냄. 만약 `None` 이면, 그래프의 모든 엣지를 대상으로 예측을 수행
- 주요 동작
  - 만약 `edge_label_index` 가 주어지지 않았으면, 그래프의 엣지 인덱스 `edge_index` 를 사용하여 `get_embedding` 메서드를 호출하여 노드의 임베딩을 얻습니다.
  - 얻은 노드 임베딩을 이용하여 `predict_link_embedding` 메서드를 호출하여 엣지의 예측값을 얻습니다.
- 참고. if isinstance(edge_index, SparseTensor):
  - 목적은 `edge_index` 를 `SparseTensor` 와 일반적인 텐서 두 가지 경우에 대해 일관된 형태로 `edge_label_index` 를 만들어주는 것
  - `edge_index` 가 `SparseTensor` 일 때:
    - `edge_index.coo()` 는 `SparseTensor` 의 좌표(coordinate) 정보를 반환합니다.
    - `edge_index.coo()[:2]` 는 좌표 정보 중에서 첫 번째와 두 번째 차원을 추출합니다.
    - `torch.stack(..., dim=0)` 는 추출한 두 개의 차원을 쌓아서 새로운 텐서를 생성합니다. 이렇게 하면 `edge_label_index` 에는 엣지의 좌표 정보가 저장됩니다.
  - `edge_index` 가 `SparseTensor` 가 아닌 경우:
    - 이 부분은 `edge_index` 가 일반적인 텐서인 경우를 다룹니다. 이 경우에는 `edge_index` 자체를 `edge_label_index` 로 사용합니다.

Code_Designing our model

```
def predict_link(self, edge_index: Adj, edge_label_index: OptTensor = None,
                 prob: bool = False) -> Tensor:

    pred = self(edge_index, edge_label_index).sigmoid()
    return pred if prob else pred.round()
```

def predict_link()

- 주어진 `edge_index` 와 `edge_label_index` 를 사용하여 엣지의 존재 여부를 예측
- 주요 구성
  - `edge_index` : 그래프의 엣지 인덱스로, 각 엣지에 대한 정보를 나타냅니다.
  - `edge_label_index` : 모델이 예측할 엣지의 인덱스를 나타냅니다.
  - `prob` : 불리언 값으로, True이면 확률값을 반환하고 False이면 0 또는 1의 이진값을 반환합니다.
- 코드 동작
  1. `self(edge_index, edge_label_index)` 를 호출하여 모델의 `forward` 메서드를 이용해 엣지의 예측값을 계산
  2. `.sigmoid()` 를 호출하여 예측값을 시그모이드 함수를 통과시켜 확률값으로 변환
  3. `prob` 이 False이기 때문에 반올림하여 0 또는 1의 이진값으로 반환
- 결과
  - 엣지의 예측값을 계산하고, 이를 통해 확률값 또는 이진값을 반환

```
def predict_link_embedding(self, embed: Adj, edge_label_index: Adj) -> Tensor:

    embed_src = embed[edge_label_index[0]]
    embed_dst = embed[edge_label_index[1]]
    return (embed_src * embed_dst).sum(dim=-1)
```

def predict_link_embedding

- 모델이 학습한 임베딩을 사용하여 엣지의 임베딩 예측값을 계산
- 주요 구성
  - `embed` : 그래프의 노드 임베딩으로, 모델이 학습한 노드 특성 벡터를 나타냅니다.
  - `edge_label_index` : 모델이 예측할 엣지의 인덱스를 나타냅니다.
- 주요 동작
  1. `embed[edge_label_index[0]]` 를 통해 `edge_label_index` 에 해당하는 엣지의 출발 노드 임베딩을 가져온다
  2. `embed[edge_label_index[1]]` 를 통해 `edge_label_index` 에 해당하는 엣지의 도착 노드 임베딩을 가져온다
  3. `(embed_src * embed_dst).sum(dim=-1)` 를 계산하여 출발 노드 임베딩과 도착 노드 임베딩의 element-wise 곱을 구하고, 마지막 차원(dim=-1)을 기준으로 합을 구한다
- 결과
  - 결과값은 엣지의 예측값을 나타내며, 두 노드 임베딩의 유사도를 나타내는 지표로 사용됩니다. 일반적으로 두 노드 임베딩이 유사할수록 예측값이 높아지게 됩니다.

Code_Designing our model

```python
def recommend(self, edge_index: Adj, src_index: OptTensor = None,
              dst_index: OptTensor = None, k: int = 1) -> Tensor:
    out_src = out_dst = self.get_embedding(edge_index)

    if src_index is not None:
        out_src = out_src[src_index]

    if dst_index is not None:
        out_dst = out_dst[dst_index]

    pred = out_src @ out_dst.t()
    top_index = pred.topk(k, dim=-1).indices

    if dst_index is not None:  # Map local top-indices to original indices.
        top_index = dst_index[top_index.view(-1)].view(*top_index.size())

    return top_index
```

def recommend()

- 모델이 학습한 노드 임베딩을 사용하여 추천을 수행
- 주요 구성
  - `edge_index` : 그래프의 엣지 인덱스로, 모델이 학습한 그래프의 구조를 나타냄
  - `src_index` : 추천을 수행할 출발 노드들의 인덱스
  - `dst_index` : 추천 대상이 될 도착 노드들의 인덱스
  - `k` : 각 출발 노드마다 추천할 도착 노드의 수.
- 주요 동작
  1. `out_src` 와 `out_dst` 에 `self.get_embedding(edge_index)` 를 통해 그래프 전체의 노드 임베딩을 가져온다
  2. `src_index` 가 주어진 경우, `out_src` 를 해당 인덱스에 해당하는 노드들의 임베딩으로 갱신
  3. `dst_index` 가 주어진 경우, `out_dst` 를 해당 인덱스에 해당하는 노드들의 임베딩으로 갱신
  4. `out_src @ out_dst.t()` 를 계산하여 각 출발 노드와 도착 노드 간의 유사도 행렬을 얻는다
  5. `pred.topk(k, dim=-1).indices` 를 통해 각 출발 노드마다 상위 k개의 도착 노드 인덱스를 얻는다
  6. `dst_index` 가 주어진 경우, 이 인덱스를 원래 인덱스로 매핑한다
- 결과
- `top_index` 에는 각 출발 노드에 대한 상위 k개의 추천 도착 노드 인덱스가 저장됨

```python
def link_pred_loss(self, pred: Tensor, edge_label: Tensor,
                   **kwargs) -> Tensor:
    loss_fn = torch.nn.BCEWithLogitsLoss(**kwargs)
    return loss_fn(pred, edge_label.to(pred.dtype))
```

def link_pred_loss()

- 모델이 예측한 엣지의 존재 여부와 실제 엣지의 레이블 간의 손실을 계산하는 함수
- `BCEWithLogitsLoss` : 이진 교차 엔트로피 손실 함수
- `pred` 는 모델이 예측한 엣지의 존재 여부
- `edge_label` 은 실제 엣지의 레이블
- 예측값과 실제 레이블 간의 손실을 계산하여 반환한다
- 이 손실은 역전파를 통해 모델의 파라미터를 조정하는 데 사용된다

Code_Designing our model

```python
def recommendation_loss(self, pos_edge_rank: Tensor, neg_edge_rank: Tensor,
                        lambda_reg: float = 1e-4, **kwargs) -> Tensor:
    r"""Computes the model loss for a ranking objective via the Bayesian
    Personalized Ranking (BPR) loss."""
    loss_fn = BPRLoss(lambda_reg, **kwargs)
    return loss_fn(pos_edge_rank, neg_edge_rank, self.embedding.weight)
```

def recommendation_loss()

- 모델의 랭킹 목표를 위한 손실을 계산하는데 사용
  - 랭킹 목표를 위한 손실: 모델이 주어진 양성 샘플(예를 들면, 실제로 사용자가 선택한 항목)의 랭킹을 올리고, 음성 샘플(사용자가 선택하지 않은 항목)의 랭킹을 낮추도록 학습하는 데 사용됨
- 사용된 손실 함수는 Bayesian Personalized Ranking (BPR) 손실
  - 주로 추천시스템에 사용되는 손실
  - BPR은 사용자가 선택한 항목과 선택하지 않은 항목 간의 상대적인 순위를 기반으로 모델을 학습시킨다
  - 주어진 양성 엣지의 예측 순위가 음성 엣지의 예측 순위보다 높도록 모델을 학습시키는데 사용
- 주요 구성
  - `pos_edge_rank` 는 양성 엣지(실제 연결된 엣지)의 순위를 나타내는 텐서
  - `neg_edge_rank` 는 음성 엣지(실제로 연결되지 않은 엣지)의 순위를 나타내는 텐서
  - `lambda_reg` 는 정규화 항에 대한 가중치

```python
def bpr_loss(self, pos_scores, neg_scores):
  return - torch.log(torch.sigmoid(pos_scores - neg_scores)).mean()
```

def bpr_loss()

- 두 가지 항목에 대한 점수 간의 차이를 이용하여 랭킹 목표를 위한 손실을 정의
- 주요 구성
  - `pos_scores` 는 양성 예제(사용자가 선택한 항목 쌍)에 대한 모델의 예측 점수
  - `neg_scores` 는 음성 예제(사용자가 선택하지 않은 항목 쌍)에 대한 모델의 예측 점수
- 정의 방법

$$\text{BPR Loss} = -\log\left(\sigma(\text{pos\_scores} - \text{neg\_scores})\right)$$

  1. 두 점수 간의 차이를 계산: `pos_scores - neg_scores`.
  2. 이 차이에 sigmoid 함수를 적용하여 두 점수 간의 상대적인 확률을 얻습니다.
  3. 얻은 확률의 로그 값을 취합니다.
  4. 최종적으로 음의 로그 우도(Negative Log Likelihood)를 계산합니다.
- 이렇게 계산된 BPR 손실은 모델이 양성 예제에 대해 높은 점수를 부여하고, 음성 예제에 대해 낮은 점수를 부여하도록 학습하는 데 도움이 됨

```python
def __repr__(self) -> str:
    return (f'{self.__class__.__name__}({self.num_nodes}, '
            f'{self.embedding_dim}, num_layers={self.num_layers})')
```

def __repr__(self)

- `__repr__` 메서드는 객체의 "official" 또는 "formal" 문자열 표현을 반환하는데 사용
- `GCN` 클래스의 `__repr__` 메서드는 해당 클래스의 이름, 노드 수, 임베딩 차원 및 레이어 수를 포함하는 문자열을 반환
- 해당 클래스의 주요 구성 요소를 간략하게 설명하여 개발자가 객체의 구성을 빠르게 파악할 수 있도록 돕는 역할

Code_Designing our model


```
class BPRLoss(_Loss):
```

```
__constants__ = ['lambda_reg']
lambda_reg: float
```

```
def __init__(self, lambda_reg: float = 0, **kwargs):
    super().__init__(None, None, "sum", **kwargs)
    self.lambda_reg = lambda_reg
```

2) Define the loss function

- Negative edge: An edge that doesn't exist in the graph
  - negative edge is one that does not exist, say *(i, k)*
- Positive edge is an existing track-playlist combination that we want to predict
  - positive edge *(i, j_+)* is just a supervision edge (an existing edge that we wish to predict) in the graph
- 개인화된 랭킹 손실로, 관찰된 항목의 예측이 관찰되지 않은 항목의 예측보다 높도록 장려한다

$$L_{\text{BPR}} = -\sum_{u=1}^{M}\sum_{i\in\mathcal{N}_u}\sum_{j\notin\mathcal{N}_u}\ln\sigma(\hat{y}_{ui}-\hat{y}_{uj})+\lambda||\mathbf{x}^{(0)}||^2$$

- M은 사용자 수를 나타냅니다.
- $\mathcal{N}_u$는 사용자 u가 평가한 항목의 집합입니다.
- $\hat{y}_{ui}$는 사용자 u가 항목 i에 대한 모델의 예측입니다.
- σ는 시그모이드 함수입니다.
- λ는 L_2 정규화 강도를 제어합니다.

 # __constants__

- 상수 속성을 정의: 여기에서는 `lambda_reg` 가 상수로 선언
- `lambda_reg` : 정규화 강도를 나타내며, 손실 함수에서 사용됨

 # `lambda_reg: float`

- `lambda_reg` 가 `float` 타입임을 선언

 # def __init__()

- `lambda_reg` 와 추가적인 `kwargs` 매개변수를 받는다
- `super().__init__(None, None, "sum", **kwargs)` : 부모 클래스(`torch.nn.modules.loss._Loss`)의 초기화 메서드를 호출하는 부분

Code_Evaluation

```
def recall_at_k(data, model, k = 300, batch_size = 64, device = None):
    with torch.no_grad():
        embeddings = model.get_embedding(data.edge_index)
        playlists_embeddings = embeddings[:n_playlists]
        tracks_embeddings = embeddings[n_playlists:]

    hits_list = []
    relevant_counts_list = []



    for batch_start in range(0, n_playlists, batch_size):
        batch_end = min(batch_start + batch_size, n_playlists)
        batch_playlists_embeddings = playlists_embeddings[batch_start:batch_end]

        # Calculate scores for all possible item pairs
        scores = torch.matmul(batch_playlists_embeddings, tracks_embeddings.t())

        # Set the scores of message passing edges to negative infinity
        mp_indices = ((data.edge_index[0] >= batch_start) & (data.edge_index[0] < batch_end)).
        scores[data.edge_index[0, mp_indices] - batch_start, data.edge_index[1, mp_indices] -

        # Find the top k highest scoring items for each playlist in the batch
        _, top_k_indices = torch.topk(scores, k, dim=1)

        # Ground truth supervision edges
        ground_truth_edges = data.edge_label_index

        # Create a mask to indicate if the top k items are in the ground truth supervision edg
        mask = torch.zeros(scores.shape, device=device, dtype=torch.bool)
        gt_indices = ((ground_truth_edges[0] >= batch_start) & (ground_truth_edges[0] < batch_
        mask[ground_truth_edges[0, gt_indices] - batch_start, ground_truth_edges[1, gt_indices

        # Check how many of the top k items are in the ground truth supervision edges
        hits = mask.gather(1, top_k_indices).sum(dim=1)
        hits_list.append(hits)

        # Calculate the total number of relevant items for each playlist in the batch
        relevant_counts = torch.bincount(ground_truth_edges[0, gt_indices] - batch_start, minl
        relevant_counts_list.append(relevant_counts)
```

1) Recall@k

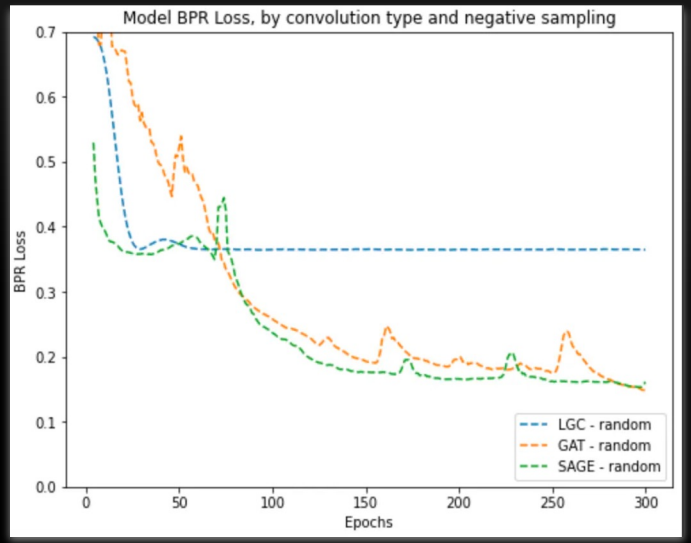
함수 초반부

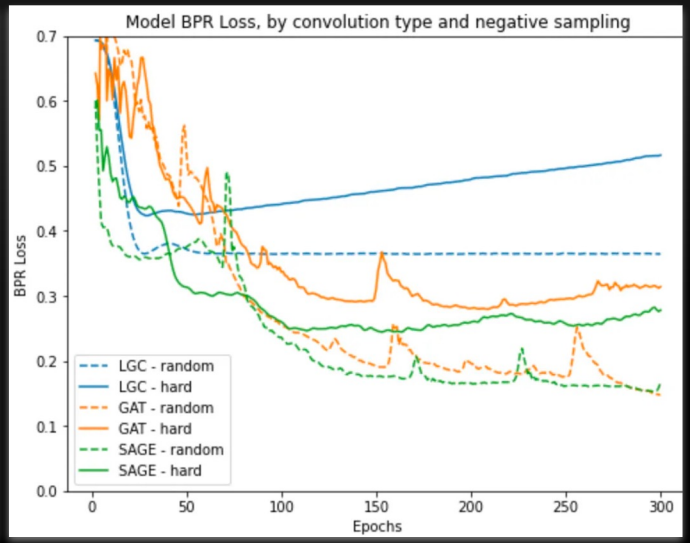
- `torch.no_grad()` 는 연산을 추적하지 않도록 하여 모델의 파라미터 업데이트를 방지한다. 이는 추론 모드에서 메모리 사용을 최적화하는 데 도움이 된다.
- `model.get_embedding(data.edge_index)` : 그래프의 엣지 인덱스에 대한 임베딩을 얻는다.
- 얻어진 임베딩을 플레이리스트와 트랙으로 나눈다.

for문

- 역할: 각 배치에 대한 Recall@k를 계산
- 주요 동작
  1. `batch_playlists_embeddings = playlists_embeddings[batch_start:batch_end]`
     - `batch_start` 에서 `batch_end` 까지의 플레이리스트 임베딩을 선택
  2. Calculate scores for all possible item pairs
     - 모든 가능한 플레이리스트와 트랙 쌍에 대한 점수를 계산합니다.
  3. Set the scores of message passing edges to negative infinity
     - 메시지 전파 엣지의 점수를 음의 무한대로 설정합니다.
  4. Find the top k highest scoring items for each playlist in the batch
     - 현재 배치에 대한 각 플레이리스트에 대해 상위 k개의 점수를 찾습니다.
  5. Ground truth supervision edges
     - 지도 엣지(ground truth supervision edges)를 가져옵니다.
  6. Create a mask to indicate if the top k items are in the ground truth supervision edges
     a. 상위 k개의 아이템이 지도 엣지에 있는지 나타내는 마스크를 만듭니다.
  7. Check how many of the top k items are in the ground truth supervision edges
     - 상위 k개의 아이템 중 몇 개가 지도 엣지에 있는지 확인합니다.
  8. Calculate the total number of relevant items for each playlist in the batch
     - 현재 배치의 각 플레이리스트에 대해 관련 항목의 총 수를 계산합니다
- 이러한 계산은 각 배치에 대해 수행되며, 결과는 `hits_list` 및 `relevant_counts_list` 에 저장된다. 이 정보는 전체 데이터셋에 대한 평균 Recall@k를 계산하기 위해 사용된다.

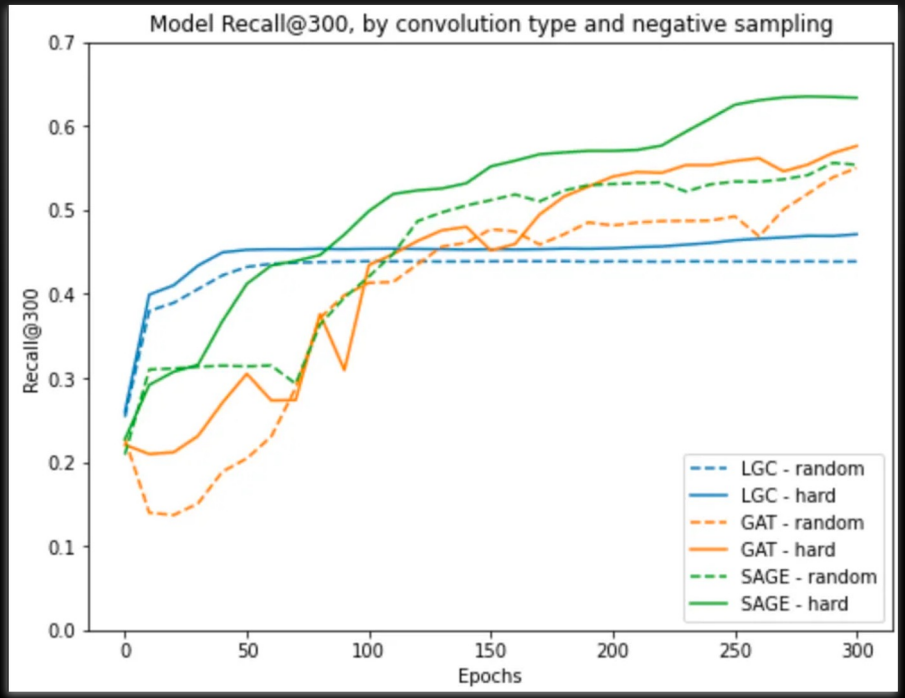
Conclusion





⇒ 전반적으로 SAGEConv가
성능, 안정성 및 수렴 속도 측면에서 가장 좋은 성능을 보임!

Conclusion

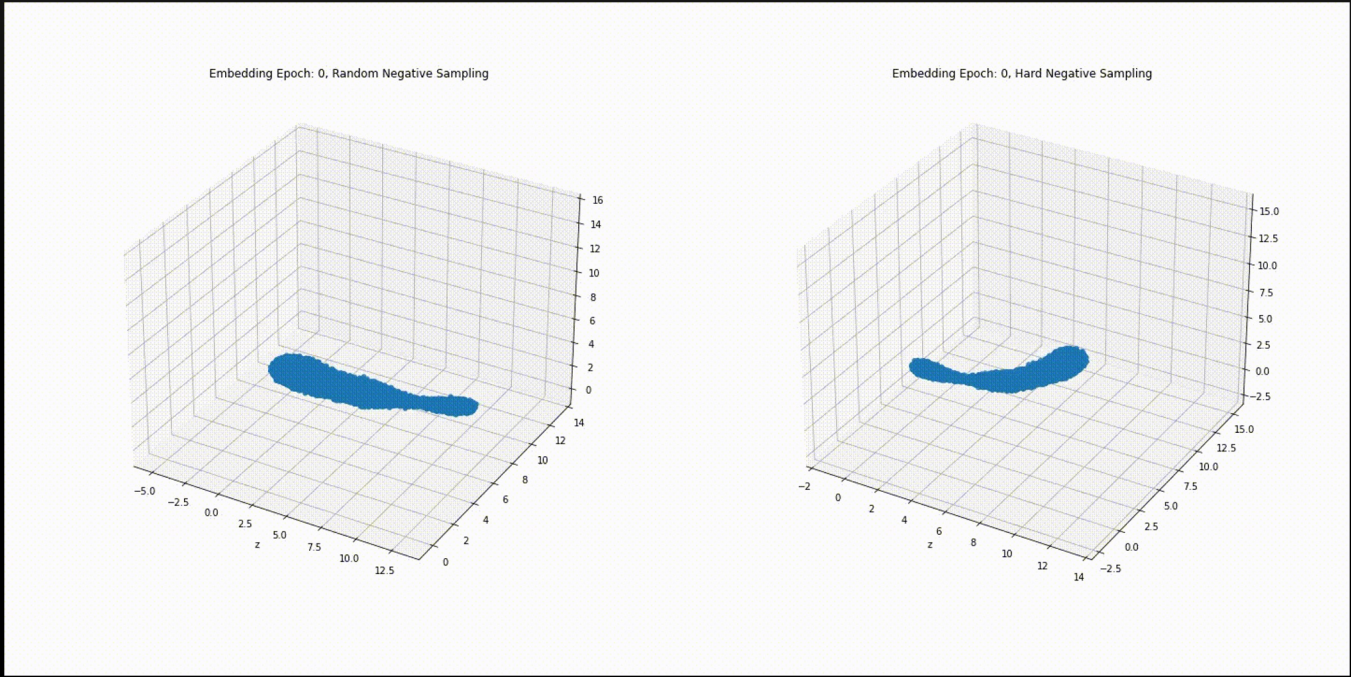


LGConv

GATConv

SAGEConv

Conclusion

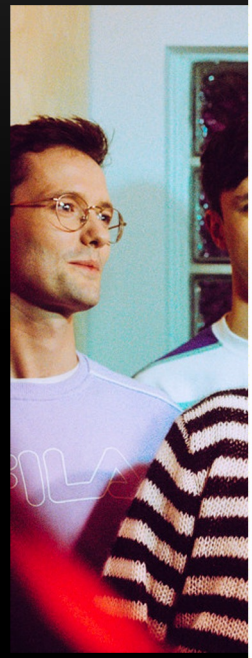


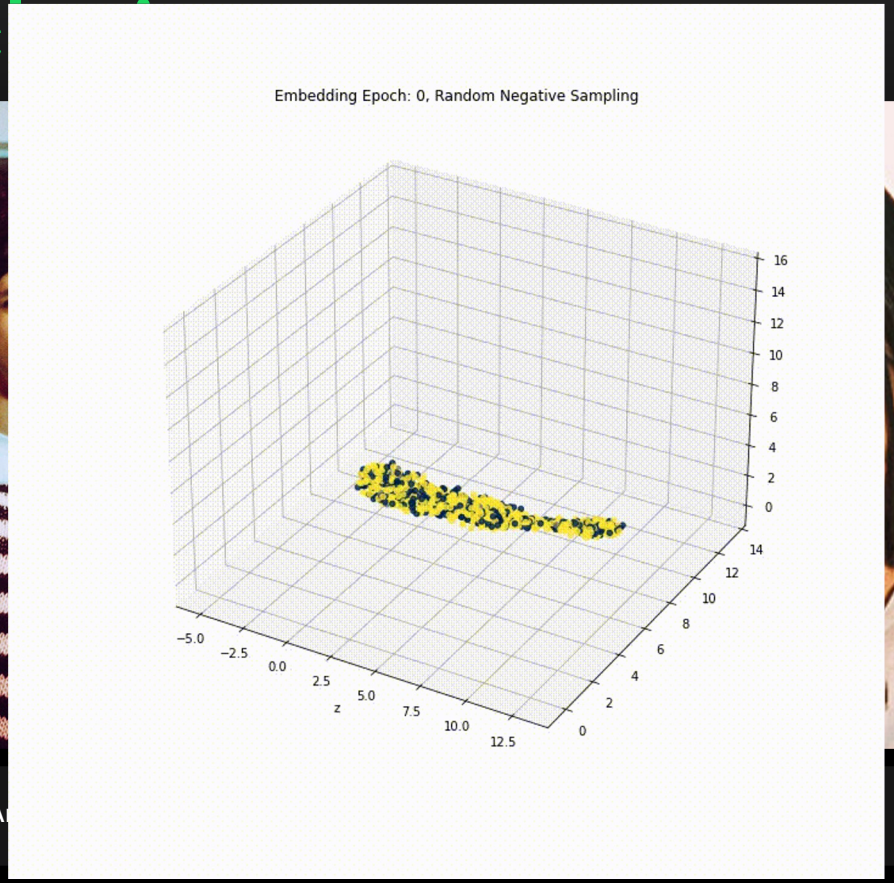
Conclusion

Glass Animals

Gryffin

Conc





Glass A

Thank You

Despacito

Luis Fonsi, Daddy Yankee

0:23 -3:25