Feature Embedding Methodology

# **0. Background**

Embedding

* 구조화 되어 있지 않은 데이터의 특징을 나타내는 벡터
* 특정 데이터를 저차원 공간으로 변환 (벡터로 매핑)
* 저차원 공간에서 유사한 데이터를 가깝게 배치함으로써 유사 데이터를 찾아낼 수 있음
* 일반적으로 비슷한 특성 값을 지닌 인스턴스는 비슷한 예측 결과로 이어지므로 feature의 표현은 학습된 패턴의 본질 및 품질에 직접적으로 영향을 줌.

**임베딩으로 표현 가능한 것(**[**Link**](https://cloud.google.com/architecture/overview-extracting-and-serving-feature-embeddings-for-machine-learning#what_is_an_embedding)**)**

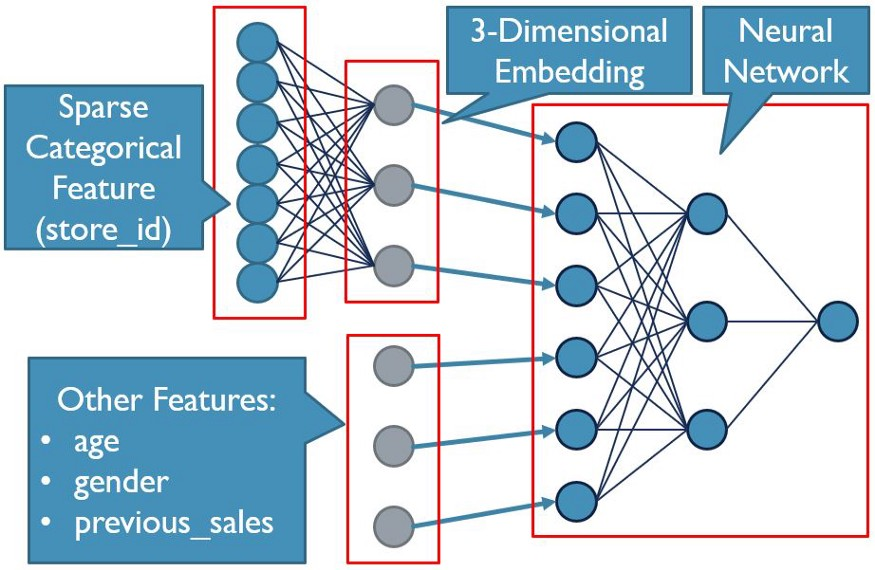
1. 텍스트, 이미지, 오디오와 같은 **구조화되지 않은 데이터**
2. 사용자가 시청한 영화목록 및 사용자 ID와 같은 **상호 작용 컨텍스트만 있는 데이터**
3. 그래프 및 네트워크와 같은 **복잡한 구조 데이터**
4. 텍스트 설명명을 통한 이미지 검색 및 이미지 캡션 작성과 같은 **다중 모달 변환**
5. 위치 및 점유와 같은 **희소 특성 → 밀집 특성**
6. 인구통계, 금융과 같은 **고차원 항목**

Transfer Learning

텍스트나 이미지, 그 외 데이터들을 인코딩하기 위해 **미리 트레이닝된 네트워크**를 이용하여 임베딩 시키는 것. **Transfer Learning**을 통해 지식을 재사용할 수 있을 뿐만 아니라, 트레이닝 시키는 시간을 대폭 줄일 수 있음

Feature Engineering

Categorical Feature를 포함하는 경우의 Embedding 예시

[](https://towardsdatascience.com/why-you-should-always-use-feature-embeddings-with-structured-datasets-7f280b40e716)

**읽어보기** <https://lsh110600.github.io/deeplearning/2021/01/10/dl-multimodal-4/>

# **Text 기반 Embedding**

## **one-hot encoding**

가장 기본적인 방법. 단어가 많아지면 벡터 공간이 매우 커지고 실제 1인 값은 한 개뿐이므로 매우 비효율적이며 값 자체의 특징을 표출해주지 못함. 벡터의 크기가 너무 크고 값이 1이 되는 값은 거의 없어 **Sparse**한 표현법이라고 한다.

Sparse한 표현법의 해결책으로 **Dense**한 표현법이 제시되었다.

이는 단어의 수에 상관없이 특정 차원의 벡터로 변환시켜 단어의 특성이나 유사성을 나타내 준다. 사이킷런의 CountVectorizer, TfidfVectorizer이나 Word2Vec이 대표적이다.

## **CountVectorizer**

각 텍스트에서 횟수를 기준으로 특징을 추출하는 방법.

단어 사전을 사전에 정의하여 문장 내 단어의 등장 횟수를 벡터화.

예를 들어 '나는 매일 일기를 쓴다'에서 단어 사전이 [나는, 매일 , 혼자, 일기를, 쓴다] 이면, [1, 2, 0, 1, 1]이 된다.

단순히 횟수만을 특징으로 잡기 때문에 큰 의미가 없고 자주 쓰이는 조사가 높은 특징 값을 가진다.

## **TfidfVectorizer**

TF-IDF를 이용해 텍스트 데이터의 특징을 추출하는 것.

TF는 Term Frequency로 특정 단어가 글 안에서 나오는 횟수를 말하며, IDF는 Inverse Document Frequency로 특정 단어가 여러 글에 얼마나 자주 나오는지 알려주는 지표의 Inverse(반대) 값을 의미.

다른 글에서 지시대명사나 조사가 많이 나오므로 IDF는 낮은 값을 갖게 되어, 의미가 없는 조사나 지시대명사를 제외한 단어들의 임베딩 값을 얻을 수 있다. TF-IDF는 TF와 IDF를 곱한 값으로 다른 글에서 자주 나오지 않고 해당 문서에 많이 등장할수록 더 높은 값을 갖게 된다.

## **Word2Vec**

<https://simpling.tistory.com/entry/Embedding-%EC%9D%B4%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%B8%EA%B0%80-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0>

# **Image Embedding**

# 이미지의 원시 픽셀(2,048 ✕ 2,048)을 다른 사진과 비교하여 비슷한지 여부를 찾는 것은 매우 비효율적. 이미지의 저차원적 특성 벡터(임베딩)를 추출하면 이미지에 포함된 내용이 무엇인지를 나타내는 일정한 지표를 얻고, 더 효과적으로 비교할 수 있다.

<https://jsideas.net/ImageEmbeddingWithTripletLoss/>

## **SBERT(**[**논문**](https://arxiv.org/pdf/1908.10084.pdf)**,** [**링크**](https://www.sbert.net/examples/applications/image-search/README.html)**)**

* + 1. Siamese Network

<https://tyami.github.io/deep%20learning/Siamese-neural-networks/>

* + 1. Triplet Network

<https://m.blog.naver.com/4u_olion/221478534498>

* + 1. BERT

<https://hwiyong.tistory.com/392>

# **Multimodal Deep Learning**

Reading List([Link](https://github.com/pliang279/awesome-multimodal-ml))

Dimensionality Reduction using AutoEncoders in Python([Link](https://www.analyticsvidhya.com/blog/2021/06/dimensionality-reduction-using-autoencoders-in-python/))

PCA vs Autoencoder([Link](https://www.researchgate.net/post/What-is-the-need-of-auto-encoder-when-we-already-have-powerful-dimension-reduction-techniques-such-as-PCA))

VAE([Link](https://www.kaggle.com/schmiddey/variational-autoencoder-with-pytorch-vs-pca/notebook))

# **Graph Embedding**

graph의 인접행렬은 크기가 매우 크므로 이를 그대로 다루기에 어려움이 있음.

-> graph의 각 node를 low-dimension으로 mapping할 필요가 있음.

즉 **Adjacency matrix의 그래프 정보를 latent dimension으로 encoding하고, node representation을 만들어 내는 것**이며, latent 에서 나타나는 node간의 similarity는 실제 graph에서의 node간의 similarity라고 볼 수 있음.

Graph Embedding은 크게 Node embedding과 Graph Embedding으로 분류.

([링크1](https://www.theteams.kr/teams/2849/post/73221), [링크2](https://dbstndi6316.tistory.com/249?category=957031))

* **Node Embedding**

그래프 하나를 평면에 시각화하거나 유사성을 사용해서 새로운 연결을 예측할 때 사용 ex) 페이스북 친구 추천

* + DeepWalk
  + Node2vec
  + SDNE
* **Graph Embedding**

그래프 전체를 벡터로 표현. 그래프 수준에서 예측, 비교, 시각화를 할 때 사용 ex) 화학 구조 간의 비교

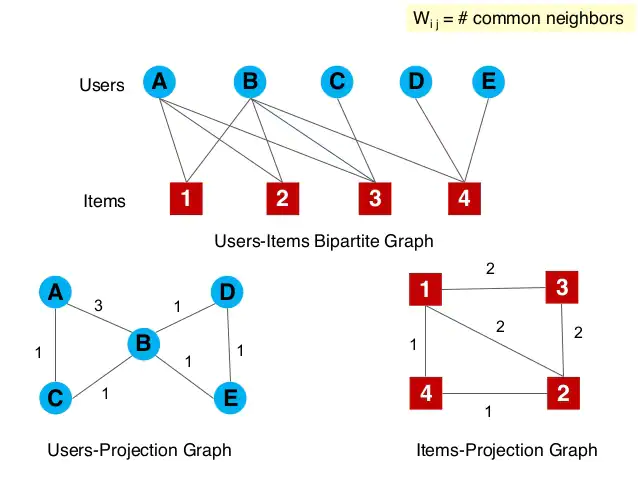
* + Graph2vec

**GCMC 적용 방법론 Idea**

기존 그래프를 임베딩 -> 의미가 있나?

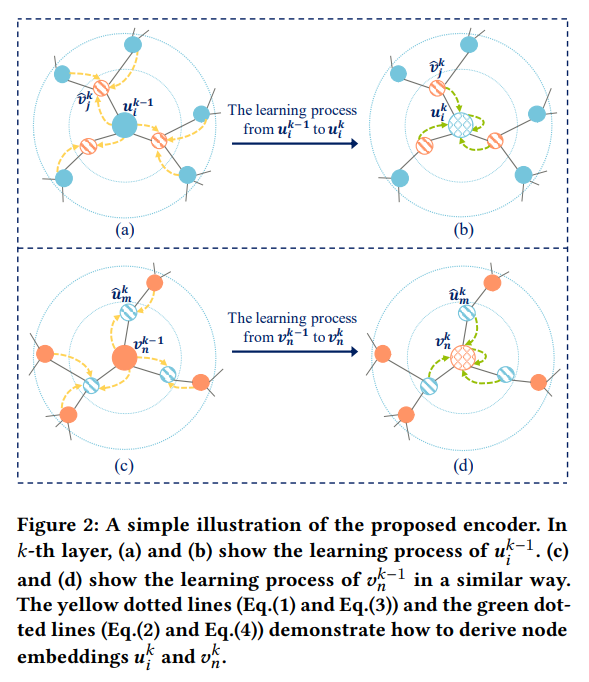
feature vector를 포함한 그래프를 임베딩

유저 간/ 아이템 간 그래프를 각각 임베딩



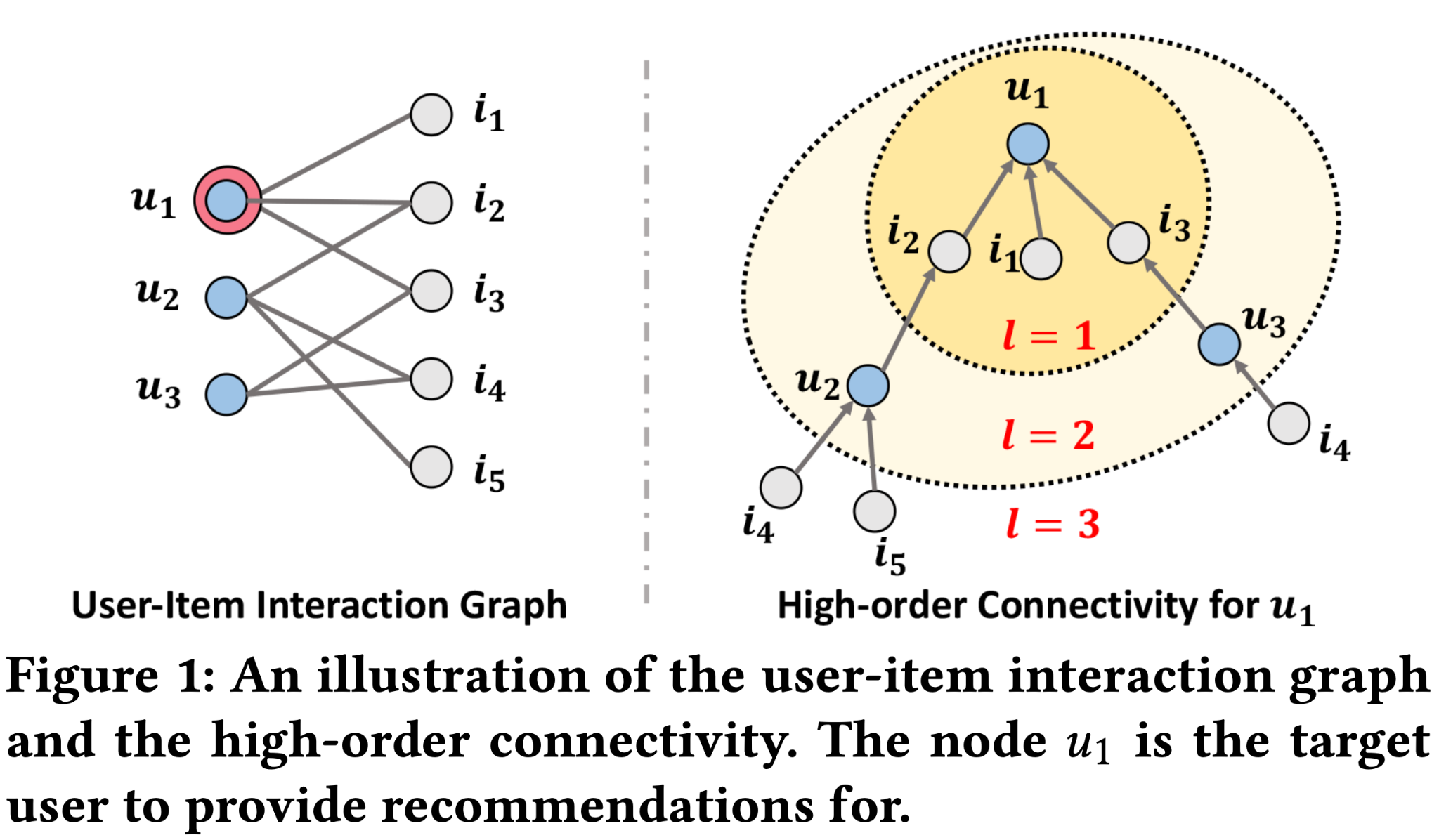
## **Bipartite Graph Embedding**

### Bipartite Graph Embedding via Mutual Information Maximization([논문](https://arxiv.org/abs/2012.05442), [리뷰](https://velog.io/@soyoungcho/Paper-Review-Bipartite-Graph-Embedding-via-Mutual-Information-Maximization))



* two-hop neighbor의 노드 임베딩을 학습해 각 노드에 대한 노드 임베딩 값을 구함
* u에 대한 representaion과 v에 대한 representation을 함께 이용해 global representation 표현
* subgraph-level에서(가까운 노드들 이용) attention weight를 구해 local representation 표현
* 이 두 representation간 mutual information을 최대화함으로 성능을 높임
* 다른 임베딩 방법론의 한계인 global properties를 모두 포착할 수 있음
  + homogenous 노드 간 community structure
  + heterogeneous와 노드 간 long-range dependencies
* Bipartite graph의 초점을 둔 방법론(PinSage, IGE, BiNE, FOBE 등) 중 state-of-the-art

### Neural Graph Collaborative Filtering(논문, [리뷰](https://velog.io/@ysn003/%EB%85%BC%EB%AC%B8-Neural-Graph-Collaborative-Filtering))



user-item interaction graph에서 high-order connectivity를 이용하는 framework