DGL on Real World Applications

Quan Gan

AWS Shanghai AI Lab

April 19, 2020

Various GNN Business Applications

- Recommender Systems: find the most relevant items for a given user.
- Product Search: find the most relevant items given a set of keywords.
- Fraud Detection: detect anomaly of entities in a supervised/unsupervised manner.
- Community Detection: cluster entities in a supervised/unsupervised manner.

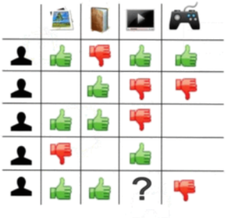
Recommender System: Problem Statement

Image source: Wikipedia

Collaborative Filtering

Collaborative Filtering

$\longrightarrow$



Collaborative Filtering

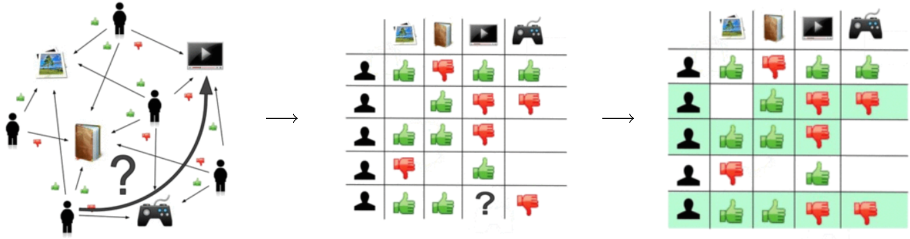
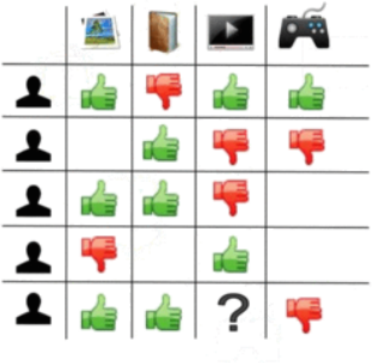


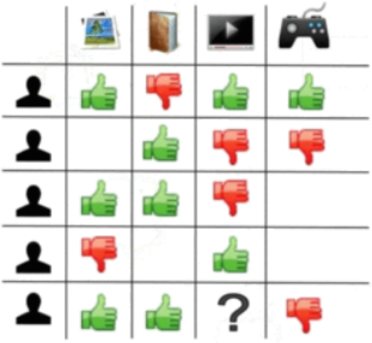
Image source: Wikipedia

Collaborative Filtering with Machine Learning

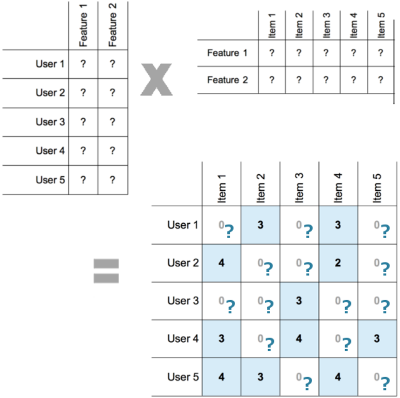


We were "representing" users and items with the items/users that had interactions with them.

Collaborative Filtering with Machine Learning



We were "representing" users and items with the items/users that had interactions with them.

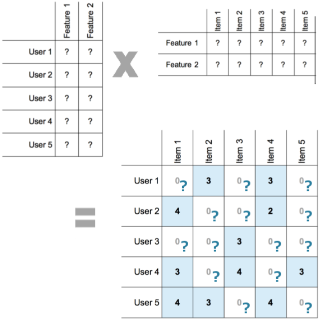


Source: Kat Bailey

Can we represent users and items as a set of features?

Latent User/Item Representations

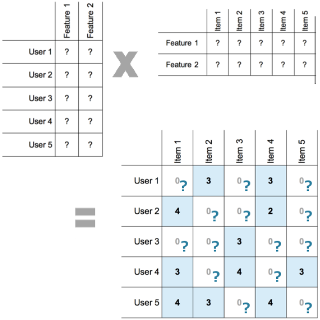
- An item can be described with a set of features (e.g. how sweet some food is).



Source: Kat Bailey

Latent User/Item Representations

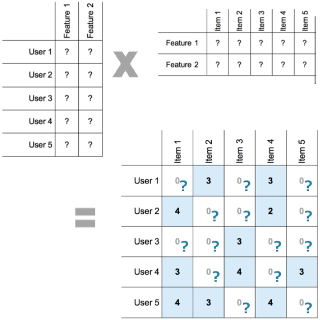
- An item can be described with a set of features (e.g. how sweet some food is).
- A user can be described with preferences of the same set of features (e.g. how much a user likes sweet food).



Source: Kat Bailey

Latent User/Item Representations

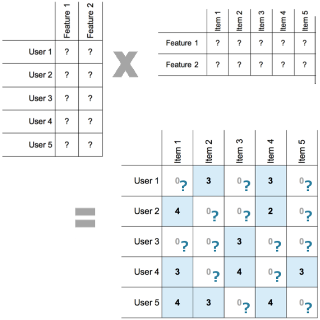
- An item can be described with a set of features (e.g. how sweet some food is).
- A user can be described with preferences of the same set of features (e.g. how much a user likes sweet food).
- The interaction is defined by how well the item features match the user preferences.



Source: Kat Bailey

Latent User/Item Representations

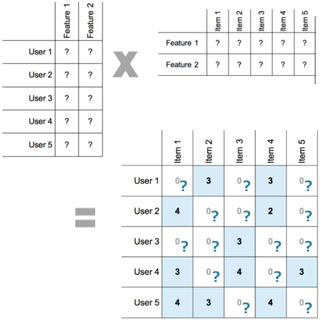
- An item can be described with a vector v_j (sweet, organic, etc.).
- A user can be described with preferences of the same set of features (e.g. how much a user likes sweet food).
- The interaction is defined by how well the item features match the user preferences.



Source: Kat Bailey

Latent User/Item Representations

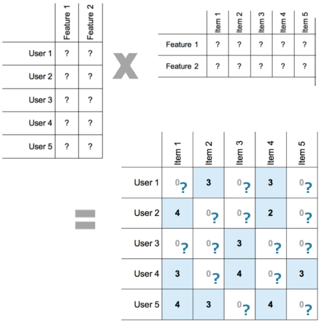
- An item can be described with a vector v_j (sweet, organic, etc.).
- A user can be described with another vector u_i (likes sweet, likes organic, etc.)
- The interaction is defined by how well the item features match the user preferences.



Source: Kat Bailey

Latent User/Item Representations

- An item can be described with a vector v_j (sweet, organic, etc.).
- A user can be described with another vector u_i (likes sweet, likes organic, etc.)
- The rating on item j by user i. is defined by $u_i^\top v_j$.

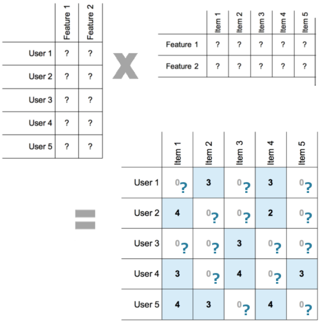


Source: Kat Bailey

Latent User/Item Representations

- An item can be described with a vector v_j (sweet, organic, etc.).
- A user can be described with another vector u_i (likes sweet, likes organic, etc.)
- The rating on item j by user i. is defined by $u_i^\top v_j$.
- We minimize

$$\sum_{i,j} \left(r_{i,j} - \left(u_i^\top v_j \right) \right)^2$$

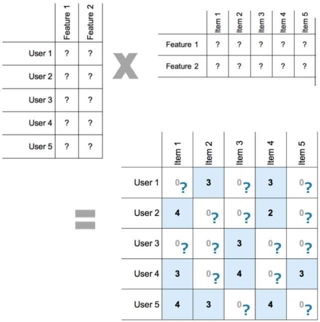


Source: Kat Bailey

Latent User/Item Representations

- An item can be described with a vector v_j (sweet, organic, etc.).
- A user can be described with another vector u_i (likes sweet, likes organic, etc.)
- The rating on item j by user i. is defined by $u_i^\top v_j$.
- We minimize

$$\sum_{i,j} \left(r_{i,j} - \left(u_i^\top v_j + b_{u_i} + b_{v_j} \right) \right)^2$$



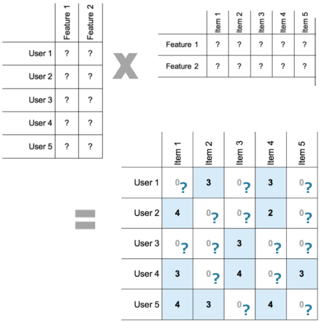
Source: Kat Bailey

Latent User/Item Representations

- An item can be described with a vector v_j (sweet, organic, etc.).
- A user can be described with another vector u_i (likes sweet, likes organic, etc.)
- The rating on item j by user i. is defined by $u_i^\top v_j$.
- We minimize

$$\sum_{i,j}\left(r_{i,j} - \left(u_i^\top v_j + b_{u_i} + b_{v_j}\right)\right)^2$$

$$+\alpha\mathcal{R}(U, V)$$



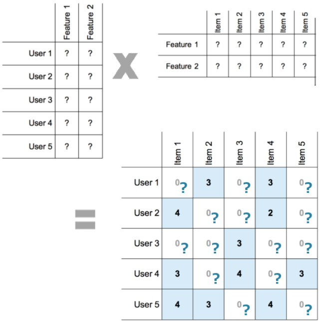
Source: Kat Bailey

Latent User/Item Representations

- An item can be described with a vector v_j (sweet, organic, etc.).
- A user can be described with another vector u_i (likes sweet, likes organic, etc.)
- The rating on item j by user i. is defined by $u_i^\top v_j$.
- We minimize

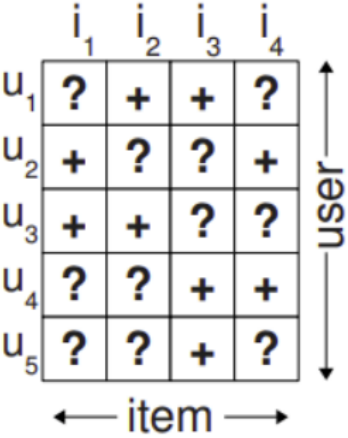
$$\sum_{(i,j)\in\mathcal{B}} \left(r_{i,j} - \left(u_i^\top v_j + b_{u_i} + b_{v_j}\right)\right)^2$$

$$+\alpha\mathcal{R}(U, V)$$



Source: Kat Bailey

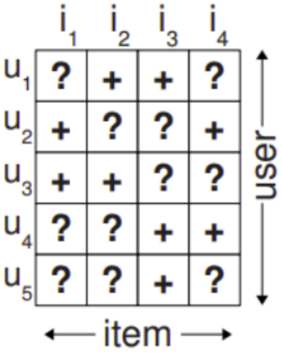
What if we don't have ratings?



Source: *BPR: Bayesian Personalized Ranking from Implicit Feedback*, Rendle et al. 2012

Implicit Feedback

- For a given user i, an item being interacted j should have a higher score than another item k which was never being interacted.

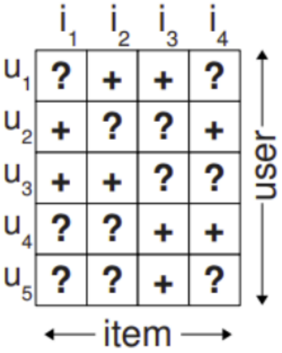


Source: *BPR: Bayesian Personalized Ranking from Implicit Feedback*, Rendle et al. 2012

Implicit Feedback

- For a given user i, an item being interacted j should have a higher score than another item k which was never being interacted.
- We maximize

$$\sum_{i,j,k \in I \setminus I_{u_i}} \log \mathrm{sigmoid}\left(\left(u_i^\top v_j - u_i^\top v_k\right)\right)$$



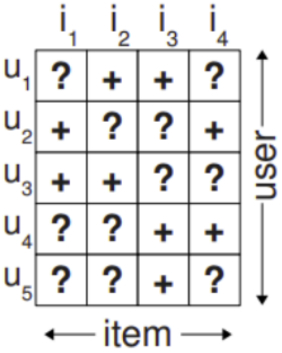
Source: *BPR: Bayesian Personalized Ranking from Implicit Feedback*, Rendle et al. 2012

Implicit Feedback

- For a given user i, an item being interacted j should have a higher score than another item k which was never being interacted.
- We maximize

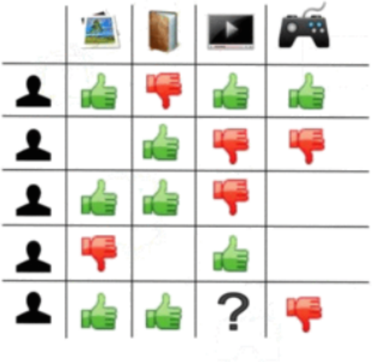
$$\sum_{i,j,k \in I \setminus I_{u_i}} \log \mathrm{sigmoid}\left(\left(u_i^\top v_j - u_i^\top v_k\right)\right)$$

- We usually *sample* one or multiple k when computing gradients (**negative sampling**).
    - Commonly uniformly, but adaptive sampling often helps.



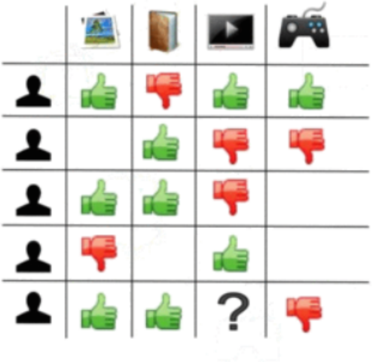
Source: *BPR: Bayesian Personalized Ranking from Implicit Feedback*, Rendle et al. 2012

Formulating Recommender Systems as Graph Learning

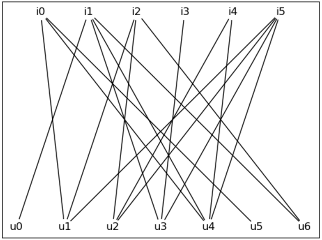


- Explicit Feedback
- Implicit Feedback

Formulating Recommender Systems as Graph Learning

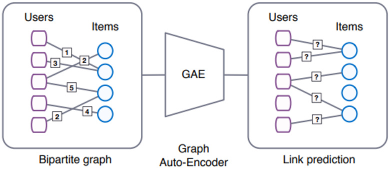


- Explicit Feedback
- Implicit Feedback



- Edge Classification/Regression
- Link Prediction

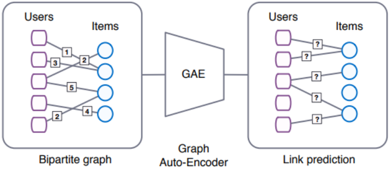
GCMC: Learning u_i and v_j from User-Item Graph



Source: *Graph Convolutional Matrix Completion*, van den Berg et al. 2017

1. $\mu_{v_j \to u_i, r} = \frac{1}{c_{u_i v_j}} W_r x_{v_j}$

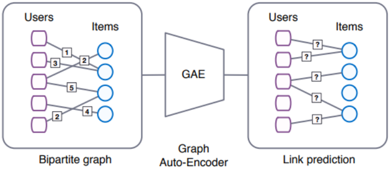
GCMC: Learning u_i and v_j from User-Item Graph



Source: *Graph Convolutional Matrix Completion*, van den Berg et al. 2017

1. $\mu_{v_j \to u_i, r} = \frac{1}{c_{u_i v_j}} W_r x_{v_j}$
2. $h_{u_i} = \sigma \left[\mathrm{agg} \left(\sum_{v_j \in \mathcal{N}_{u_i,1}} \mu_{v_j \to u_i, 1}, \cdots, \sum_{v_j \in \mathcal{N}_{u_i,R}} \mu_{v_j \to u_i, R} \right) \right]$

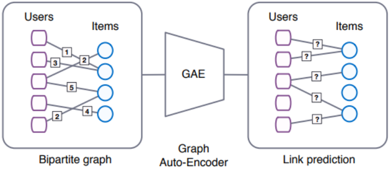
GCMC: Learning u_i and v_j from User-Item Graph



Source: *Graph Convolutional Matrix Completion*, van den Berg et al. 2017

1. $\mu_{v_j \to u_i, r} = \frac{1}{c_{u_i v_j}} W_r x_{v_j}$
2. $h_{u_i} = \sigma \left[\mathrm{agg} \left(\sum_{v_j \in \mathcal{N}_{u_i,1}} \mu_{v_j \to u_i, 1}, \cdots, \sum_{v_j \in \mathcal{N}_{u_i,R}} \mu_{v_j \to u_i, R} \right) \right]$
3. $u_i = \sigma(W_u h_{u_i})$ and similarly we compute v_j

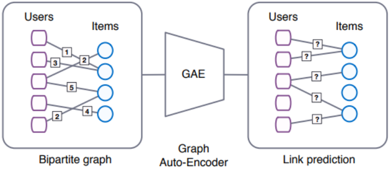
GCMC: Learning u_i and v_j from User-Item Graph



Source: *Graph Convolutional Matrix Completion*, van den Berg et al. 2017

1. $\mu_{v_j \to u_i, r} = \frac{1}{c_{u_i v_j}} W_r x_{v_j}$
2. $h_{u_i} = \sigma \left[\mathrm{agg} \left(\sum_{v_j \in \mathcal{N}_{u_i,1}} \mu_{v_j \to u_i, 1}, \cdots, \sum_{v_j \in \mathcal{N}_{u_i,R}} \mu_{v_j \to u_i, R} \right) \right]$
3. $u_i = \sigma(W_u h_{u_i})$ and similarly we compute v_j
4. $p(\hat{M}_{ij} = r) = \mathrm{softmax}(u_i^\top Q_r v_j)$

GCMC: Learning u_i and v_j from User-Item Graph



Source: *Graph Convolutional Matrix Completion*, van den Berg et al. 2017

1. $\mu_{v_j \to u_i, r} = \frac{1}{c_{u_i v_j}} W_r x_{v_j}$
2. $h_{u_i} = \sigma \left[\mathrm{agg} \left(\sum_{v_j \in \mathcal{N}_{u_i,1}} \mu_{v_j \to u_i, 1}, \cdots, \sum_{v_j \in \mathcal{N}_{u_i,R}} \mu_{v_j \to u_i, R} \right) \right]$
3. $u_i = \sigma(W_u h_{u_i})$ and similarly we compute v_j
4. $p(\hat{M}_{ij} = r) = \mathrm{softmax}(u_i^\top Q_r v_j)$
5. If users and items have no features, we make x_{v_j} and x_{u_i} themselves learnable parameters.

Other Meaningful Aspects to Consider

- **Cold-start**: What if we have *new* users and items coming in, with few to no historical interactions and features?

Other Meaningful Aspects to Consider

- **Cold-start**: What if we have *new* users and items coming in, with few to no historical interactions and features?
- **Bias correction**: The training dataset usually comes from the result of a *previous recommender system*. How to mitigate the bias?

Other Meaningful Aspects to Consider

- **Cold-start**: What if we have *new* users and items coming in, with few to no historical interactions and features?
- **Bias correction**: The training dataset usually comes from the result of a *previous recommender system*. How to mitigate the bias?
- **Diversity**: Always recommending the same items (or even the same kind of item) to a user would make him/her feel *bored*.

Other Meaningful Aspects to Consider

- **Cold-start**: What if we have *new* users and items coming in, with few to no historical interactions and features?
- **Bias correction**: The training dataset usually comes from the result of a *previous recommender system*. How to mitigate the bias?
- **Diversity**: Always recommending the same items (or even the same kind of item) to a user would make him/her feel *bored*.
- **Fraud**: How to detect and deal with fabricated explicit feedbacks (e.g. fake ratings and reviews)?

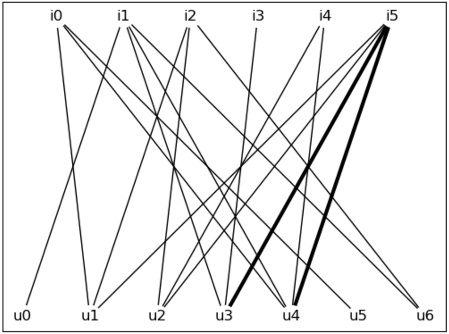
Non-exhaustive List of Recommender System Models

- PinSAGE
- STAR-GCN
- Knowledge Graph Attention Networks
- Knoledge Graph Convolutional Networks
- Inductive Matrix Completion (IGMC)
- ...

Hands-on Session

Miniature GCMC on bipartite user-item graph with DGL.

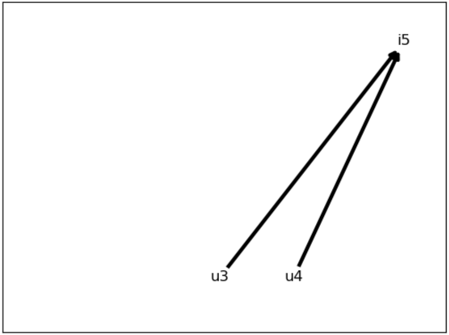
How to Train GCMC Stochastically?



- Given the graph above, we wish to predict the rating of the selected edge minibatch with a 1-layer GCMC.

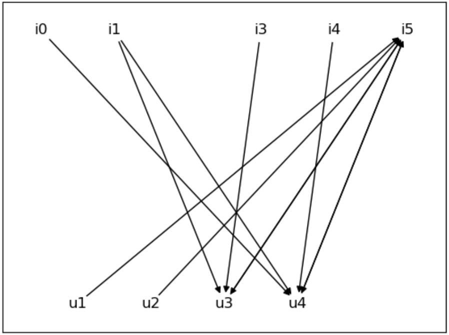
Constructing a "Pair Graph"

- DGL provides an `apply_edges` method that computes edge features in parallel based on source, destination, and other edge features.
- Construct a pair graph that consists of the edges to be trained only.



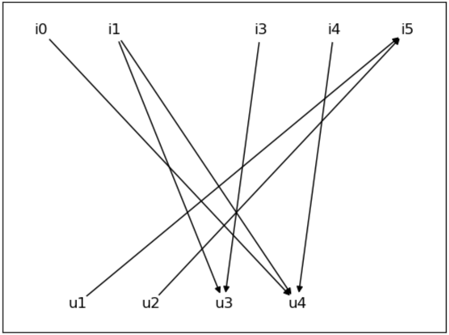
Find Necessary Neighbors

- The computation dependency can be used in message passing by transforming the dependency graph by `dgl.to_block` into a bipartite-structured graph as explained in the previous tutorial.



Removing Information Leakage

- Since we are training to predict the rating of `(u3, i5)` and `(u4, i5)`, we don't want to tell the model that these edges exist. Therefore we need to remove these edges.



The Computation Procedure (Reference to Notebook)

- Sample a minibatch of edges.
- Construct a pair graph from those edges.
- Construct the computation dependency graph.
- Transform the dependency graph with `dgl.to_block` for message passing.
- Copy features from the original graph to the blocks.
- Compute the outputs from the GNN layers on the blocks.
- Copy the output of the GNN layers to the pair graph.
- Compute the score on the edges of the pair graph with `dgl.apply_edges`.