Nathan Tsai & Abdullatif Jarkas

Professors Mishne & Fraenkel

DSC 180B

7 March 2021

Project Report

Graph-Based Product Recommendation

Abstract

Recommender systems are important, revenue-generating technologies in many of the services today, providing recommendations for social, product, and other networks. However, the majority of existing recommender system methods use metrics of similarity to recommend other nodes through content-based and collaborative filtering approaches, which do not take into account the graph structure of the relationships between the nodes. A graph-based recommender system then is able to utilize graph relationships to improve node embeddings for recommendation in a way that conventional recommender systems cannot. Inspired by GraphSAGE[3] and PinSage[1], we explore two unsupervised graph-based approaches on the Amazon-Electronics dataset that can utilize the graph relationships of product and user data in order to generate accurate and robust embeddings for product recommendation.

Introduction

Recommender systems are responsible for large revenues and consumer satisfaction in many of the services used today. Widely-used services, such as Netflix, Facebook, Amazon, and LinkedIn, use recommender systems to suggest movies, posts, users, and products to their consumers. Traditional recommender system methods use metrics of similarity to recommend

other products through content-based and collaborative filtering approaches. However, product data can be expressed in a non-Euclidean graph format with relationships, such as products bought together or products viewed together. Traditional recommender system methods do not take into account the graph relationships between the product nodes in the same way graph neural networks do. A graph-based approach to product recommendation is able to fully utilize the relationships between product nodes to produce improved product embeddings that can represent the graph structures of product data.

Motivation

Recommender systems play a large role in suggesting products in online marketplaces or connections on social media networks. High quality recommendations are important to increase consumer satisfaction and boost sales revenues and user interactions. The motivation behind our project is to apply graph neural networks to the complex and important task of recommender systems. Though traditional recommender system approaches take into account product features and user reviews, traditional methods do not address the inherent graph structure between products and users or between products themselves.

Related Work

Our work builds upon the existing advancements in applying graph neural networks to recommender systems. Graph convolutional networks (GCNs)[2] have allowed deep learning to harness the power of non-Euclidean data, applying deep learning techniques to graph relationship and structure data. GraphSAGE[3] introduced an inductive approach to generating embeddings that sampled neighboring nodes and aggregated their features to produce

embeddings. PinSage[1] improved upon the GraphSAGE algorithm by introducing a graph-based recommender system with a new sampling and aggregation process and providing an efficient technique for large, web-scale training for production models at Pinterest. We adapt the GraphSAGE and PinSage algorithms to work in an unsupervised learning context to generate Amazon product embeddings, which take into account the underlying graph structure.

Data

We use the Amazon Electronics datasets from Professor Julian McAuley. We use the Electronics product metadata dataset, Electronics 5-core reviews dataset, and the processed Electronics product image features. The 5-core reviews dataset represents a subset of Electronics product reviews, in which every product has at least 5 product reviews and every reviewer has written at least 5 reviews. Each product has a variety of features like price, title, description, sales rank, brand, product categories, an image that has been passed through a deep convolutional neural network to produce image embedding vectors consisting of 4096 floats[4]. Each review has features like overall rating, a helpfulness ratio, product review, product review summary, and the review time.

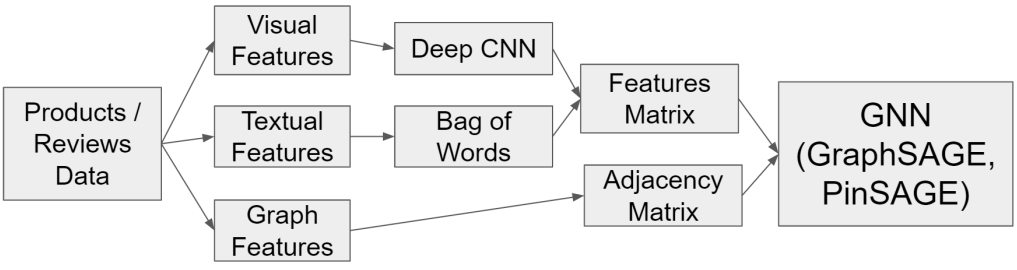


Figure 1. Overview of Model and Data Pipeline.

GraphSAGE Data Preprocessing

In the product metadata dataset, the products are identified by their unique Amazon standard identification numbers, or ASINs. We first dropped all rows with missing data. Then we filtered out our data to keep products that have related products specifically ‘also_bought’, which reveals all ASINs of products that were purchased along that specific product. These ‘also_bought’ products describe our target nodes in our edge list. After that, we removed all ‘also_bought’ products that do not exist in our dataset. Then through the category column we manipulate it into another column to get its sub-category or specified niche to then have a label for our product nodes. Because our ASINs/Product IDs are strings, we cannot enter it into our model, so then we mapped all ASINs in our dataset to a unique integer. Then we formed our edge list. After that we used a TF-IDF Vectorizer to vectorize the title text features of the product to serve as the sole product node feature. After extracting our node labels (niche category), node features (product title), and edge list, we began constructing our graph, and then inputting the graph and graph features into our model.

We then created a graph of all nodes and edges and then divided the train and test sets based by edges. We did a 90-10 split where 10% of the edges were located in the test graph and 90% of the edges were located in the training graph. We did this so our model can learn features relationships based on similar and not identical ones.

GraphSAGE Graph

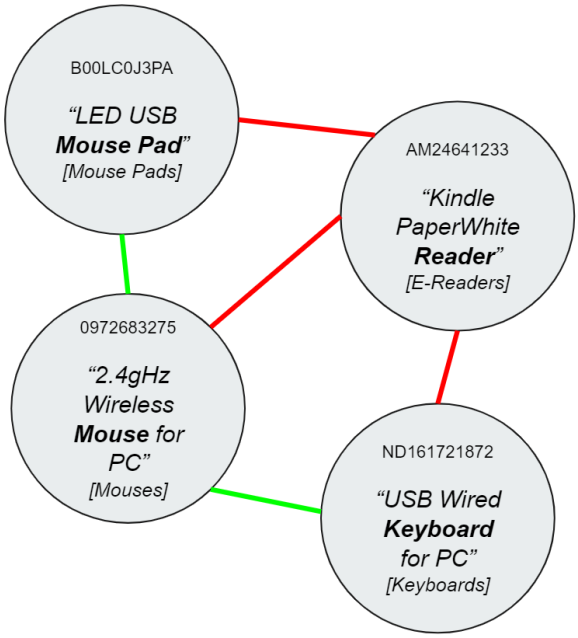


Figure 2. Diagram of Product Graph for GraphSAGE.

Our GraphSage graph is a homogenous graph consisting of products as nodes and edges connected on whether those nodes were purchased together. With 19,532 nodes and 430,411 edges we had a lot to work with. Each node contained product title text features and also a label, it's specific category. We split the edges into two, positive and negative edges. The positive edges (Green links in Figure 2) describe actual co-purchased relationships between two products and the negative edges (Red links in Figure 2) is just the inverse, describing absolutely no relationship.

Pinsage Data Preprocessing

In the reviews dataset, products are identified by their unique ASINs, and users are identified by their unique 'reviewerID'. We drop the 'reviewerName', 'reviewTime', and

'reviewText' columns, then dropping all rows with missing data. We process the 'helpful' column by dividing the helpful votes by the overall votes to get a helpfulness ratio, filling in 0.0 if there are no overall votes. Using the product ASINs found in our reviews dataframe, we read in the image features dataset, only keeping the product ASINs that appear in our reviews dataframe, creating a final list of product ASINs with both reviews and image features. We then read in the product metadata and filter both the product metadata and reviews with the final product ASIN list. Missing product prices are imputed using the median price, multiplied by 100, and converted to integers. The dataset is split into 80:10:10 ratios for train, validation, and test datasets. Using review time as the index, the model is trained on the older review events to predict 'future' review events. The training split is converted into a graph, and the validation and test splits are converted into user-product adjacency matrices.

PinSage Graph

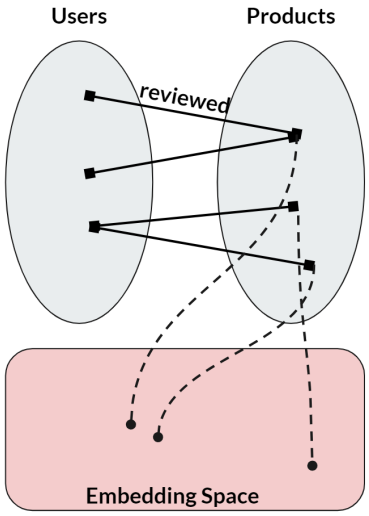


Figure 3. Diagram of Bipartite User-Product Graph for PinSage.

Using products represented by unique ASINs and users represented by unique reviewer IDs, we build a bipartite, heterogeneous graph, using reviews as undirected edges between users and products. For the products, we use the product price and image representations as features. For the review edges, we add overall rating, helpfulness, and review time as features. We did not have any user features to use. The full graph has 62,521 product nodes, 192,403 user nodes, and 1,674,664 review edges. The training graph has 62,521 product nodes, 192,403 user nodes, and 1,289,858 review edges. The validation and test matrices have 192,403 edges each.

Methods

GraphSAGE Model

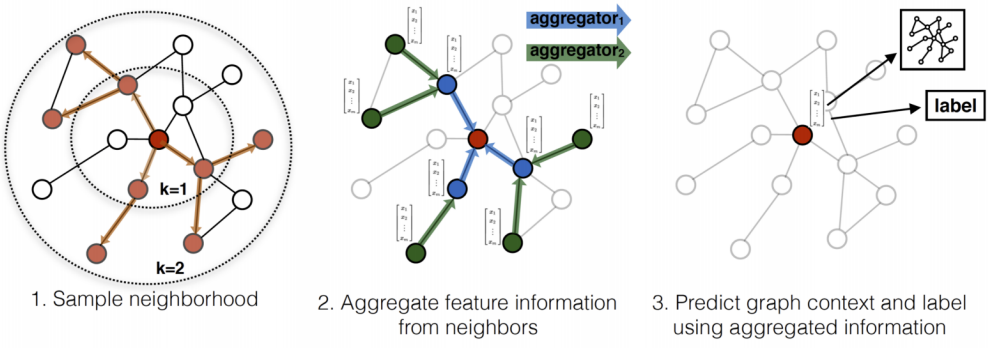


Figure 4. Diagram of GraphSAGE Algorithm

The GraphSAGE Model is a slight twist on the graph convolutional model. GraphSAGE samples a target node's neighbors and their neighboring features and then aggregates them all together to learn and hopefully predict the features of the target node. Our GraphSAGE model works solely on the node feature (product title), and the relationships through the edges of co-purchased products.

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Figure 5. GraphSAGE Embedding Algorithm

Our GraphSAGE model consisted of two layers, each aggregating neighbors based on mean. At each step of our model, our model learns through calculating the dot product of both the source and target node and then applying that feature onto an edge. As we have both positive and negative edges describing true and false co-purchased relationships between products, we compared both edges to our model through a Dot Predictor to then calculated our total loss through binary cross entropy loss.

PinSage Model

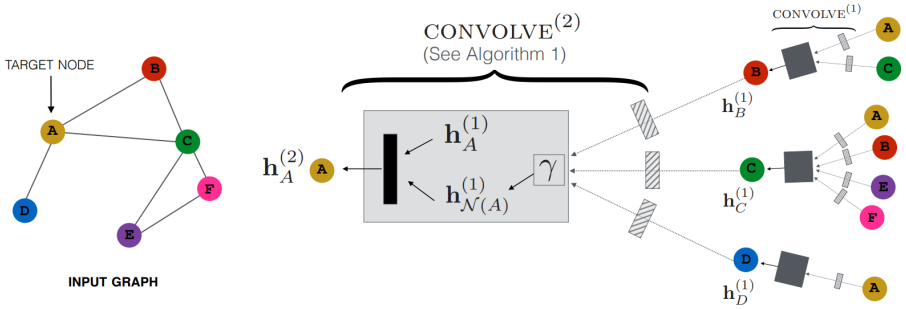


Figure 6. Diagram of PinSage Algorithm

The second model is based on PinSage[1] and learns to generate node embeddings using visual features, textual features, other product features, and graph features. Visual features are

image embeddings generated by passing product images through a deep convolutional neural network[4]. Textual features are processed using Pytorch's TorchText library to create vocabularies used in training Bag of Words models. The PinSage model takes in a heterogeneous, bipartite graph connecting user nodes to product nodes through reviews. Similar to the GraphSAGE algorithm, the PinSage model aggregates neighboring node features using an element-wise aggregation function, concatenates the aggregated neighborhood representation to the node's current representation before passing the entire vector through a neural network layer and normalized to get a final representation. This final node embedding represents both the node itself and its local graph neighborhood.

Input : Current embedding $\mathbf{z}_u$ for node u; set of neighbor embeddings $\{\mathbf{z}_v | v \in \mathcal{N}(u)\}$, set of neighbor weights $\boldsymbol{\alpha}$; symmetric vector function $\gamma(\cdot)$

Output: New embedding $\mathbf{z}_u^{\text{NEW}}$ for node u

$\mathbf{n}_u \leftarrow \gamma\left(\{\text{ReLU}\left(\mathbf{Q}\mathbf{h}_v + \mathbf{q}\right) \mid v \in \mathcal{N}(u)\}, \boldsymbol{\alpha}\right)$;

$\mathbf{z}_u^{\text{NEW}} \leftarrow \text{ReLU}\left(\mathbf{W} \cdot \text{CONCAT}(\mathbf{z}_u, \mathbf{n}_u) + \mathbf{w}\right)$;

$\mathbf{z}_u^{\text{NEW}} \leftarrow \mathbf{z}_u^{\text{NEW}} / \|\mathbf{z}_u^{\text{NEW}}\|_2$

Figure 7. PinSage Embedding Algorithm

Unlike the GraphSAGE algorithm, PinSage determines which neighboring nodes are best used for aggregation for the best representation. The PinSage sampler constructs node neighborhoods by performing multiple random walks, keeping track of which neighboring nodes are visited most often. The most visited neighbors are considered more important for aggregation. This method allows the model to aggregate neighboring nodes based on a factor of importance in an aggregation process called 'importance pooling'.

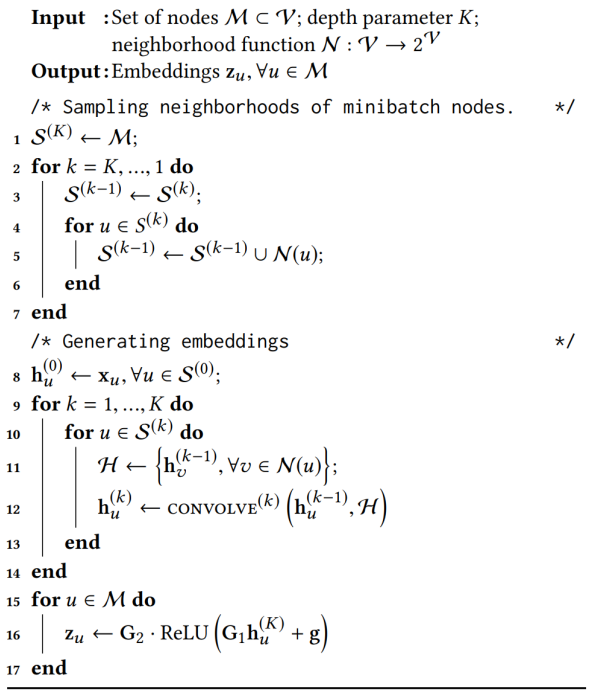


Figure 8. PinSage Sampling and Embedding Algorithm

Though the PinSage model is a supervised model based on Pinterest data with ground truth edge labels, we adapted our model to an unsupervised context, where the model predicts whether two products are reviewed by the same user. Additionally, we add product node and user node IDs as features to create learnable embeddings for each node. However, this makes the model transductive and can be excluded for inductive applications.

We train the PinSage model using 10 random walks of length 2 with a restart probability of 50%. The model aggregates the top 3 neighboring nodes, from 2 different SAGE convolutional layers, to generate 64-length product embeddings. The model was first trained for 200 epochs of 1024 batches per epoch at 32 batch size and 3e-5 learning rate. Then it was trained for 40 epochs of 5000 batches per epoch at 64 batch size and 5e-4 learning rate.

Nearest Neighbor Product Recommendation

By combining the embedding approaches of visual, textual, and graph features, our graph-based approaches can generate a more complete and robust embedding of each product. To generate recommendations given a product, the product will first be fed through the model to generate an embedding. Using K-nearest neighbor methods, we find the closest embedded nodes based on Euclidean distance. The closest node embeddings to the given product embedding are the most related products. The K-closest node indices are then converted into product ASINs for recommendation.

Results

To evaluate the product recommendations of our models, we used a metric called hit rate for the top-K recommended products using K = 500. Given an unseen product, the GraphSAGE model generates the top-K closest embedded product nodes to recommend. If at least one of the recommended products is connected to the unseen product, it is considered a hit. We calculate the hit rate at K as the number of hits over the number of unseen test products. Similarly, the PinSage model generates the top-K closest embedded product nodes to recommend for each user. If at least one of the recommended products is reviewed by the user in the test matrix, it is considered a hit. We calculate the hit rate at K as the number of hits over the number of users.

Model	Hit Rate	Loss
GraphSAGE	0.6590	0.1209
PinSage	0.1563	0.2965

Table 1. Model Evaluation Results

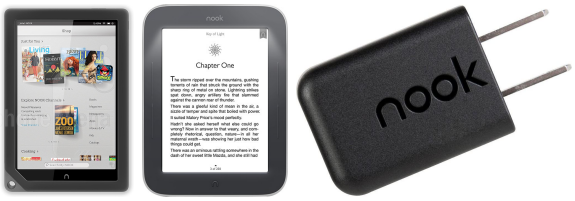
Test Product	GraphSAGE Model Recommendations
 E-Reader	
 Wireless Internet Range Extender	
 Lithium Ion Battery	

Table 2. Sample GraphSAGE Recommendations

Products User Reviewed	PinSage Model Recommendations
	
	

Table 3. Sample PinSage Recommendations

Discussion

Given the metrics of hit rate and loss, the unsupervised GraphSAGE model slightly performed better than the unsupervised PinSage model. Though the results are not 'state of the art', our results show us that product recommendation can be accomplished using graph neural networks. A reason the GraphSAGE model performs better than the PinSage model can be explained by the different product graphs used. Products bought together may be more helpful in product recommendation than products reviewed by the same user because a user may review many, completely different products that may not be helpful for product recommendation.

Conclusion

We showed how graph convolutional networks can be used in a recommender system context and utilize graph data in addition to the features considered by traditional approaches to product recommendation. Using the Amazon Electronics datasets, we explored two different graphs and graph neural networks that can be used for product recommendation. In one method, we constructed a homogenous product graph with edges between products bought together and used an unsupervised GraphSAGE model. In another method, we constructed a bipartite, heterogeneous user-product graph with edges between users and products they reviewed using an unsupervised PinSage model. The GraphSAGE model slightly outperformed the PinSage model,

but both models can be improved upon, as we did not achieve the results we had hoped for. Future considerations may be to collect user features or utilize more product features and review features, such as product description, review text, product brand, product category. Another consideration is to utilize both product-product edges and user-product edges in a unified graph neural network model.

References

[1] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, J. Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems.

[2] T. N. Kipf and M. Welling. 2017. Semi-supervised Classification with Graph Convolutional Networks.

[3] W. L. Hamilton, R. Ying, and J. Leskovec. 2017. Inductive Representation Learning on Large Graphs.

[4] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. 2015. Image-based Recommendations on Styles and Substitutes.

Appendix

Project Proposal

Problem & Data

This project is to examine recommender systems using a graph-based learning approach. Recommender systems are responsible for large revenues and consumer satisfaction in many of the services used today. Widely-used services, such as Netflix, Facebook, Amazon, and LinkedIn. use recommender systems to suggest movies, posts, users, and products to their

consumers. This project focuses on the application of graph learning to augment product recommendation. Current recommender system methods use metrics of similarity to recommend other products through content-based and collaborative filtering methods. However, product data can be expressed in a non-Euclidean graph format with relationships, such as products bought together or products in similar categories. These recommender system methods do not take into account the relationships between the product nodes, not utilizing the graph structure of data to improve recommendations. Graph-based learning can take advantage of the relationships between product nodes, in addition to the product text and image features, and generate more accurate and robust embeddings. A graph-based recommender system then is able to utilize improved product embeddings to recommend products, utilizing graph data in a way that conventional recommender systems cannot.

The dataset used in the project will be the Amazon Product Reviews dataset from Professor Julian McAuley (https://nijianmo.github.io/amazon/index.html). Each product has image and CNN features that can be used as node features, and edges are provided or can be generated using spectral clustering techniques. Edges between product nodes are expressed by products that were also viewed or bought by users. The image data can also be passed through a CNN to generate an image embedding that can also be used as a feature in graph learning.

Once we've developed a graph recommender algorithm for Amazon products we may potentially get into graph classifications concentrating on the Amazon user and their features. For example, our dataset contains product review data which contains data on both the product and the user. We could use this data with minor data ingestion and processing techniques to create nodes on users with node features such as products reviewed, products

viewed, and etc. We can use this data to find and categorize different users by their niche interests through these different node features. In this case we'd be creating an algorithm similar to graph classifications on social media networks, but on Amazon customers and their products. Which would be a unique approach to user graph classifications.

Methods

The images are provided within the dataset as URLs, the descriptions and other text features are also provided in the dataset, and the graph relationships between nodes will be constructed from the provided IDs of products also bought or viewed by similar users. We plan to utilize a convolutional neural network to generate an image feature embedding from the product image, a neural network model like word2vec to generate a text feature embedding, and a graph neural network like graphSAGE or node2vec to generate graph embeddings. By combining all three embedding approaches, we can get a more complete and robust embedding of the product for recommendation. The closest embeddings to the generated embedding will be the most related products and be recommended by model.

Project Output

The output of the project will be a report that describes how graph-based learning approaches can be used with recommender systems and include comparisons with the various recommender system approaches on benchmark datasets. The report will include examples of the recommendations across models given the same product input. An optional website output can focus on returning the images of the top recommended products within the dataset based on product information that the user submits.