Article Title

Frederik Valdemar Schrøder, Jens Petur Tróndarson, Mathias Møller Lybech

Aalborg University

fschra16@student.aau.dk jtrand16@student.aau.dk mlybec16@student.aau.dk

November 26, 2020

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

I. Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit [6].

II. Basic Theory

When dealing with systems with a lot of users and a lot of content it can be beneficial to try and help users find content that they would like. This is usually done through recommender systems. There are broadly speaking two approaches to recommender systems which are content-based filtering and collaborative filtering. Content-based filtering is done by recommending items that are similar to items that the user has liked before. This approach uses features that have to be hand engineered by categorizing the items. Collaborative filtering on the other hand uses both relationship between users and the inter dependencies among items to provide recommendations [3]. This means that collaborative filtering recommends an item to a user based on a similar user.

The advantage with collaborative filtering is that the features are learned automatically and it does not rely on hand engineered features which makes it more scalable for larger domains with many users and items. These features are called latent features. In general, collaborative filtering is more accurate than content-based filtering [3]. But collaborative filtering suffers from the cold start problem which means that it does not deal well with new users that have not rated any items yet.

i. Matrix factorization

The data concerning users and items can be represented as a matrix where each row represents a user and each column an item. The entries in the matrix will then be the explicit feedback given for that item by the user if there is any. This will be quite a sparse matrix in most cases. Matrix factorization can then be used to infer user preference for the items not rated by the user by using implicit feedback which can be purchase history, browsing history, mouse movement or any behavioral patterns that the user has. We can estimate if a user is going to like an item by analyzing the behavior of the users [3]. We start out with a matrix $A = NxM$ where N is the number of users and M is the number of items. The implicit feedback is used to learn the latent features of the items and the users liking of those latent features. This is done by mapping users and items to a joint latent factor space of dimensionality f. The items will then be associated with a vector $q_i \epsilon \mathbb{R}_f$ and the user will be associated with a vector $p_u \epsilon \mathbb{R}_f$. The matrix q_i has the dimensions kxm where k is the number of latent features and m is the number of items. Matrix q_u has the dimensions nxk where n is the number of users and k is the number of latent features. For each item i the corresponding entry in q_i will measure the extent in which the item possesses the feature and for each user u the corresponding entry in q_u will measure their liking to each feature. By then calculating the dot product $q_i^T p_u$ the resulting matrix will capture an estimate of what each user would rate each item. The main challenge of using the matrix factorization model is finding a good algorithm for computing the mapping of each item to the factor vectors q_i and q_u. After the factor vectors are computed the task of estimating the rating a user will give an item is easy through the following equation $r_{ui} = q_i^T p_u$ where r_{ui} is the resulting ratings matrix.

ii. Nerual Networks

Neural networks are a multi-layered collection of nodes which have an input layer, hidden layers, and an output layer [4]. The input of each node is the output of all nodes in the previous layer. Each of these connections in the network has an individual weight associated with it. To get the value of a node not in the input layer, the output of all the nodes in the previous layer are added togehther with each individual weight along the connection that it travels. All these weighted values are then fed to an aggregation function which result is given to an activation function. The activation function is usually the Sigmoid, Sign, or Relu function.
Neural networks can have any number of nodes in the input layer, hidden layers, and output layer and they do not need to be the same amount. Furthermore, it can have any number of hidden layers.
Neural networks have had considerable success in low-level reasoning where lots of training data are available. They learn by giving the input layer values and then have the network compute an output value. The ouput value is then compared to the real value of the inputs and based on the margin of error we go backwards through the network adjusting each weight. This is called back propagation. After doing this enough times or when the margin of error is lower than a predefined value the network is considered trained and can be tested on new data or be applied to a real-world scenario.

iii. Graph Convolutional Network

Graph Convolutional Network (GCN) is a neural network architecture that operates on graphs. Given a graph $G = (V, E)$ a GCN takes the input of a adjacency matrix A with size of NxN that represents graph G and a feature matrix NxF^0, where N is the total amount of nodes and F^0 is the total amount of input features for each node. A hidden layer in a GCN can be defined as $H^i = f(H^{i-1}, A)$ where i indicates the layer and H^0 is the previously mentioned NxF^0 feature matrix and f is a propagation function [2]. There are many different types of propagation functions. A simple example could be $f(H^i, A) = \sigma(AH^iW^i) = H^{i+1}$ where W^i is the weight matrix at layer i and σ is a non-linear activation function [2]. The intuition behind this propagation function is that the future representation of each node is calculated based on its neighbors nodes. Because of this, each time i is increased, a node's new value is therefore affected not only by its neighbors but its neighbors' neighbors and so on. An issue with this propagation function could be that the value of each node now is a sum of each of its neighbors, and therefore loses its own value. This could be solved by replacing A with $\hat{A} = A + I$ where I is the identity matrix. Doing this the node considers itself a neighbor.

III. Related work

i. Light GCN

In the paper *LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation* Xiangnan et al. investigate the effect of feature transformation and nonlinear activation within collaborative filtering using Neural Graph Collaborative Filtering (NGCF) [1]. NGCF is a framework developed by Wang et al. [5] that utilizes Graph Neural Network with three components in the framework: (1) an embedding layer that constructs initial user - and item embeddings; (2) embedding propagation layers that captures CF with the two operations of message construction and message aggregation; (3) a prediction layer that concatenates the embeddings at each layer for each user and for each item such that $e_u^{(*)} = e_u^{(0)}||...||e_u^{(l)}$ and $e_i^{(*)} = e_i^{(0)}||...||e_i^{(l)}$ where u indicates the user, i indicates the item, e is the embedding and l is the layer. Within the second component of message construction the message embedding is implemented as:

$$m_{u \leftarrow i} = \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}(W_1 e_i + W_2(e_i \bigodot e_u)), \tag{1}$$

where W_1 and $W_2 \in R^{d' \times d}$ are weight matrices and d' is the transformation size. $\frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}$ is the graph Laplacian norm, where $\mathcal{N}_u$ and $\mathcal{N}_i$ are the set of neighbors of user u and item i. This is done with the purpose of calculating how much the item contributes to the users preference [5]. LightGCN criticize the use of the weight matrices W_1 and W_2 as not being useful for CF as each user-item interaction graph only has the ID as input and it has no semantic value [1]. Also within the second component the message aggregation is implemented as:

$$e_u^{(l)} = \text{LeakyReLU}(m_{u \leftarrow u}^{(l)} + \sum_{i \in \mathcal{N}} m_{u \leftarrow i}^{(l)}), \tag{2}$$

where $e_u^{(l)}$ are the embeddings of user u at layer l. LeakyReLU is the activation function chosen in NGCF to allow encoding positive and small negative signals. LightGCN shows that NGCF will perform better if the feature transformation is removed, and that the activation function has small effect when the feature transformation is included, but if feature transformation is disabled the activation function will have a negative impact on performance. Light GCN also shows that NGCF will significantly improve if both the activation function and feature transformation are removed [1].

ii. Price-aware Recommendation with Graph Convolutional Networks

Yu Zheng et al. develops an effective method called *Price-aware User Preference-modeling (PUP)* that is used to create recommendations with focus on the price factor [**Priceaware**]. There are two main challenges. The first is that the preferences of a user on item price are unknown, and can only be seen implicitly by previous purchases. The second challenge is that the price of an item largely depends on what category the item is within. For the first problem they propose a model that creates a relationship between user-to-item and item-to-price using a Graph Convolutional Network, so that they can propagate the influence of price onto the users. For the second problem they solve this by integrating the categories and items into the propagation process. PUP represents the data as a heterogeneous undirected graph $G = (V, E)$ where the nodes in V consist of user nodes $u \in U$, item nodes $i \in I$, category nodes $c \in \mathbf{c}$, and price nodes $p \in \mathbf{p}$. The edges in E consist of interaction edges (u, i) with $R_{ui} = 1$ if there is an interaction between user u and item i and category edges $(i, \mathbf{c}_i)$ and price edges $(i, \mathbf{p}_i)$. Traditional Latent Factor Models such as Matrix Factorization only take a single type of edge (u, i) into account. This makes them insufficient when multiple types such as (u, p) and (i, c) are introduced [**Priceaware**]. Hereby Yu Zheng et al. use a Graph Neural Network to learn the embeddings so that each node has a separate embedding $e' \in \mathbb{R}^d$ where d is the dimensions of the embeddings. In GCN the nodes propagate its nearest neighbors, which could be user-item, item-price or item-category. Embeddings from node j to node i are propagated as follows:

$$t_{ji} = \frac{1}{|\mathcal{N}_i|} e'_j, \tag{3}$$

where $\mathcal{N}_i$ is the set of neighbors for node i and e'_j is the embedding node j. The updating rule is:

$$\begin{aligned}
o_u &= \sum_{j \in \{i \text{ with } R_{ui}=1\} \cup \{u\}} t_{ju} \\
o_i &= \sum_{j \in \{u \text{ with } R_{ui}=1\} \cup \{i, \mathbf{c}_i \mathbf{p}_i\}} t_{ji} \\
o_c &= \sum_{j \in \{i \text{ with } \mathbf{c}_i = c\} \cup \{c\}} t_{jc} \\
o_p &= \sum_{j \in \{i \text{ with } \mathbf{p}_i = c\} \cup \{p\}} t_{jp} \\
e_f &= \tanh(o_f), f \in \{u, i, c, p\},
\end{aligned}$$

where e_u, e_i, e_c, and e_p are the embeddings for user u, item i, category c and price p. The intuition about this updating rule is that the node gets aggregated together with its neighbors and in each layer it will go one step further. So for example a price nodes embedding will be aggregated together with all item node embeddings connected with this specific price. By doing this items with the same price levels are expected to be more similar than items not in the same price level. The intuition about the category is the same as with the price. The final purchase prediction s between user u and item i is formulated as follows:

$$s = e_u^T e_i + e_u^T e_p + e_i^T e_p + \alpha(e_u^T e_c + e_u^T e_p + e_c^T e_p), \tag{4}$$

where α is a hyper-parameter used to balance the categories influence on the prediction. The results of a Top-K Recommendation Performance test showcase that PUP outperforms other state-of-the-art methods such as NGCF with an average improvement of 3.59 % to 5.97 % [**Priceaware**].

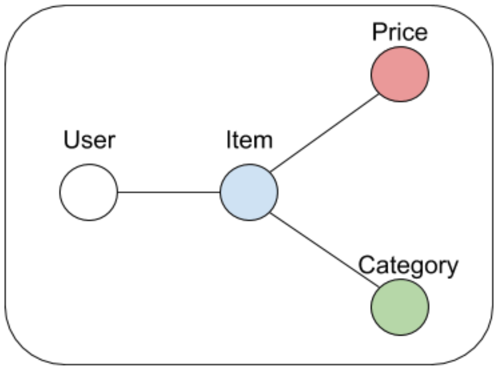

Figure 1: *Illustration of the nodes in the simple extension of LightGCN.*

IV. Method

i. Simple price aware extension of LightGCN

We try to extend the implementation of LightGCN by changing the input parameters, where we construct the adjacency matrix containing the users, item, category and price graph, which is illustrated in Figure 1. The intuition behind the idea, is that the graph convolutions will capture the values of categories and price, so even if we do not use the embeddings for price and category, they will still influence users and items. Let the user-item, item-price and item-category interactions matrix be $R \in \mathbb{R}^{I \times U+C+P}$, where I denotes the number of items, U, C, P denotes the number of users, categories and prices. Each entry of R_{ui} is 1 if $user u$ has rated $item i$. Otherwise it is 0. If there is a connection in R_{ic} or R_{ip} this value is a hyperparameter X with a value $x > 0$, otherwise it is 0. The adjacency matrix is obtained as follows:

$$A = \begin{bmatrix} 0 & R \\ R^T & 0 \end{bmatrix} \tag{5}$$

The embeddings for users and price are calculated as follows,

$$E^{(k+1)} = (D^{\frac{1}{2}} A D^{\frac{1}{2}} E^{(k)}), \tag{6}$$

where A is the adjacency matrix containing users, items, categories and price, and D is a $(I + U + C + P)$ diagonal matrix, where D_{ii} denotes the sum of the $i - th$ row in the adjacency matrix A. The $0th$ layer embedding $E^{(0)} \in R^{(I+U+C+P)\times T}$, where T is the embedding size. We do not change anything else in LightGCN in this method.

V. Experiment

i. Equal data

When measuring the performance of the different methods it is essential that the datasets are the same. Without them being equal the results of the experiments wont be worth comparing. Both PUP and LightGCN utilized the yelp dataset according to their papers. We did not get the dataset from the PUP implementation and were unable to find the original yelp-2018 dataset from the LightGCN implementation were the price and category was still attached. We therefore decided to utilize the yelp-2020 dataset, in which users rank businesses between 1 to 5 starts. Inspired by the PUP method we also decided to only use businesses with the restaurant category. What is left is set of businesses with one or more subcategories that relate to what type of restaurant they are. Like PUP we cut of any extra subcategories so that each business now only has one subcategory. A future way to further extend PUP could be by having more than one subcategory for each item. The single subcategory that an item possesses is referred to as its category in the rest of this paper.
PUP has a 60% training-, 20% validation- and 20% test-data split while LightGCN has a 70% training-, 10% validation-, and 20% test-data split. It was decided to use the LightGCN split since this was the method we planned to extend.
After this we now have a dataset that can be used to compare the different methods with or without category and price.

VI. Discussion

VII. Conclusion

References

[1] Xiangnan He et al. "LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation". In: SIGIR '20. 2020.

[2] Tobias Skovgaard Jepsen. *How to do Deep Learning on Graphs with Graph Convolutional Networks*. 2018. URL: `https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780` (visited on 10/14/2020).

[3] Y. Koren, R. Bell, and C. Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8 (2009), pp. 30–37.

[4] David L. Poole and Alan K. Mackworth. "Artificial Intelligence: Foundations of computational agents (second edition)". In: ed. by Cambridge University Press 2017. 2017.

[5] Xiang Wang et al. "Neural Graph Collaborative Filtering". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2019).

[6] Rex Ying et al. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems". In: KDD '18. 2018.