TOP-K ITEM RECOMMENDATIONS FOR CONTENT CURATION PLATFORMS USING A GRAPH CONVOLUTIONAL AUTOENCODER

An Undergraduate Research Scholars Thesis

by

CHARLES IM

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. James Caverlee

May 2022

Major: Computer Engineering

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Charles Im, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Top-K Item Recommendations for Content Curation Platforms Using a Graph Convolutional Autoencoder

Charles Im
Department of Computer Science and Engineering
Texas A&M University

Research Faculty Advisor: Dr. James Caverlee
Department of Computer Science and Engineering
Texas A&M University

There are millions of users using applications where content creators (curators) create items or content for the users to consume. With users spending more and more time on these platforms, personalized recommendations incentivizes users to consume more content. Much research has been done to improve the performance of the systems but the process of using a graph-based relationship to map the users and curators is still a largely new topic that has the potential to capture the relationships between users, content, and curators. In this thesis, we propose a graph-based convolutional autoencoder recommender system for top-K item recommendations for each user. We compare the results of our model to current state of the art recommender systems and offer insight into the noise that impacts the relationship between users and curators. We demonstrate that our model performs similarly to current state of the art models and provide future directions that require more research.

DEDICATION

To our families, instructors, and peers who supported us throughout the research process.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. James Caverlee, and Jianling Wang, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

The data analyzed/used for Top-K Item Recommendations for Content Curation Platforms Using a Graph Convolutional Autoencoder were provided by the TAMU InfoLab. The analyses depicted in Top-K Item Recommendations for Content Curation Platforms Using a Graph Convolutional Autoencoder were conducted in part by the TAMU InfoLab and these data are unpublished.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Undergraduate research was supported by the Undergraduate Research Scholars thesis program at Texas A&M University.

This work received no funding.

1. INTRODUCTION

With the increasing amount of consumable content accessible to people, there is a greater reliance in using recommender systems to connect users to the right content across many different platforms. Recommender systems allow users to receive more personalized content, so that users can consume more items that they prefer, which leads to increased activity on those platforms. Usually, recommender system models are trained by using data of users' past actions and applying an algorithm to predict items that different users may be interested in consuming. Improvements in recommender systems have been shown to positively influence user engagement in different platforms [1, 2]. Recommender systems, however, are not limited to only recommending items to users. Recommender systems have a wide spectrum of applications in different fields which extend the possibilities of recommending various types of items to people [1–3].

One exciting direction in recommender system research is the field of curation platforms. Research in traditional recommender systems for users and items has been done extensively [4–6], but research involving interactions between users and *content creators* has not been conducted as extensively due it to being a relatively new area. Platforms like Spotify implement features that allow users to listen to other curators' playlists, while users of platforms like YouTube and TikTok directly rely on the consumption of content made by other curators. All of the respective recommender systems on the aforementioned platforms play an integral role in each of their functionalities and are needed to maintain or increase user engagement. As a result, further research into the interactions between users, curators, and items should be done to have a greater understanding in providing *curator-based* recommendations.

In content curation platforms, users tend to follow curators who create content that are similar in preference to users' interests. Creating state of the art recommender systems allows those platforms to recommend content that users are more likely to consume [2, 7, 8]. To illustrate, the popular platform, YouTube, lets users "subscribe" to content curators and uses users' subscriptions and watch histories to provide new video recommendations. The existence of content curators allows users to more easily access specific content that they enjoy and allows the platform to organize a system that uses user-curator and user-item interactions to provide more accurate recommendations. Such a system creates a framework for recommender system specialists to utilize the overarching structure to generate user-specific recommendations.

An important issue in curator-content based recommendation is finding a way to accurately capture the interactions between users, items, and curators. User preferences are often complex and driven by a multitude of factors, so they usually cannot be confined to one certain area or genre. No single user will have the exact same preferences as another and it is not guaranteed that a user will only consume content in a singular domain. Likewise, the content that curators produce usually spans different domains and genres. When presented with user-item interactions along with user-curator interactions, the given data can be very noisy since not all user preferences align with a single curator. Therefore, it is important to capture the extent of the similarities users share with a set of curators in order to use user-curator interaction data efficiently.

Another interesting issue we want to address is whether information can be propagated from user-item to user-curator interactions. Users usually have a set of preferences and may share similar preferences with other users – preferences that are not defined in existing user-curator interactions. Since it is possible for a single user to have similar tastes with another user but not be following or interacting with them, it is worthwhile to investigate whether those similar users can be made into artificial curators, a term we call pseudo-curators. If successful, discovering those interactions could result in augmenting data about user preferences and allowing for more precise recommendations.

In this thesis, we propose to tackle the challenges mentioned above and produce a novel top-k item recommendations for users using a graph convolutional based autoencoder model on an implicit feedback dataset.

In order to capture the interactions between users and curators, we propose a graph structure to connect relevant users. A graph structure explicitly connects users to the curators they follow, while also providing a defined structure to organize user, item, and curator interactions. Utilizing the graph structure, we further adopt a graph convolutional preprocessing step to redefine user embeddings using information about the curators that they follow. To further reduce the noise in the user-curator relationship, we propose a similarity computation between the users and curators with a final autoencoder architecture to generate the actual top-k recommendations.

In addition to the graph convolution step, we explore the results of creating pseudo-curators by doing random walks of user-item interactions. We investigate making the number of pseudo-curators that have similar preferences to users proportional to the total number of the curators they follow. Furthermore, we observe that our graph convolution step alters the user embeddings that were previously expressed with binary numbers, so we add ceiling and floor thresholds as a method of normalizing the data into a format that was more representative of an implicit feedback dataset.

We show that utilizing the interactions between users and curators without a denoising agent results in decreasing performances, while the inclusion of a similarity calculation results in more accurate results.

2. BACKGROUND INFORMATION

The following sections introduce background information that is necessary to understand the context behind our proposed model. We first describe the intuition behind basic recommender systems and the impact they have on current state of the art models. The second section explains the popular autoencoder architecture and the reasons for why it is used for our particular model. The last section describes the difference between explicit and implicit feedback datasets and explains the format that our dataset is expressed in.

2.1 Intuition of Basic Types of Recommender Systems

Recommender systems are a broad field, spanning over several different domains and also having a variety of different implementations. Unlike current state of the art models, non-machine learning algorithms like content based recommender systems [5] and collaborative filtering [4] serve as the basis of the model we propose and are still being used as a basis for current learning-based recommender systems. This section will describe the intuition behind content-based recommender systems and collaborative filtering and explain how they shaped current learning-based models.

Basic content-based recommender systems utilize a single user's past interactions with items to generate other item recommendations [5]. Usually, these recommender systems use a similarity calculation between a user and the items in the dataset and ranks the items with the highest similarity value to provide recommendations that the user is mostly likely to consume. Generally, content-based recommender systems create recommendations specific to a single user and do not require other users' data.

Collaborative filtering algorithms take a user's past interactions with items and also factors in other users who share similar item preferences to generate recommendations [4]. These algorithms usually try to find patterns among different users, based on the intuition that the behavior of a single user will resemble the behaviors of other users with similar user-item interactions.

State of the art recommender systems that are used today are largely based off of the two main types of recommender systems that were described. Indeed, many current recommender models utilize hybrid models that use deep learning techniques combined with content-based and/or collaborative filtering recommender systems [1]. The combination of matrix factorization techniques with machine learning models' ability to learn complex non-linear interactions have led to the emergence of hybrid recommender systems that outperform bare-bones content-based and collaborative filtering recommender systems.

2.2 Introduction of the Autoencoder

In this section, we explore the autoencoder deep learning architecture, which we utilize in our proposed model. The goal of an autoencoder is to capture hidden interactions of a high dimension input by projecting it to a latent space and redefining the original vector [6, 9, 10].

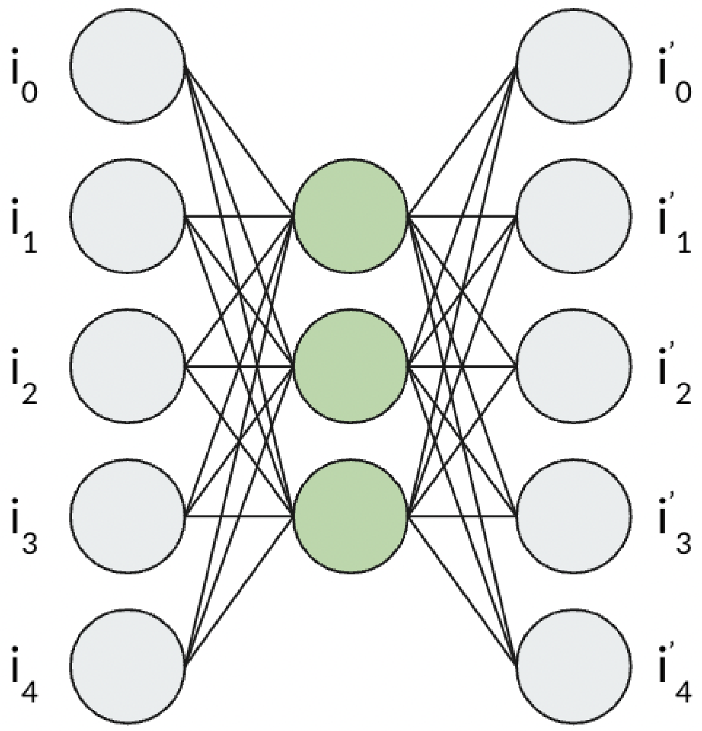


Figure 2.1: Autoencoder architecture.

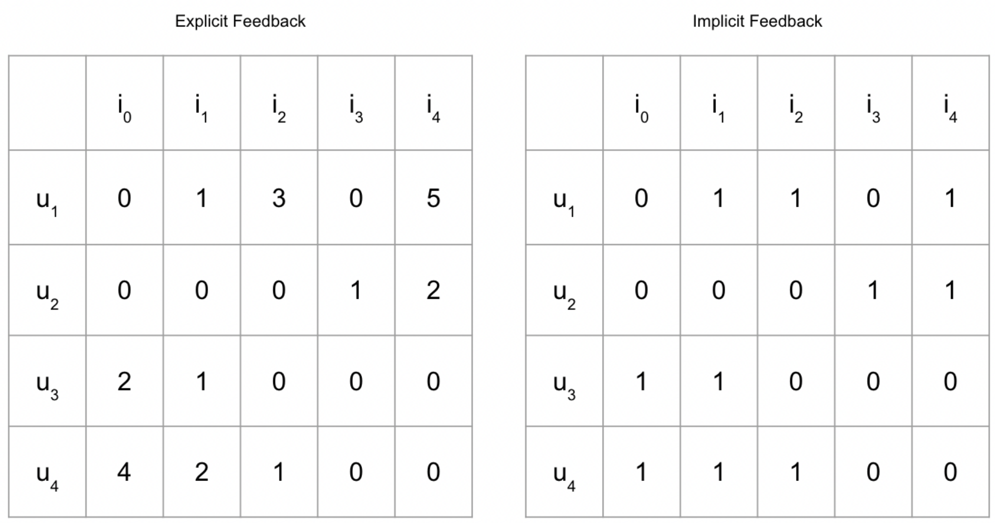
The autoencoder is a basic neural network with one hidden layer, consisting of an encoder and decoder. Figure 2.1 displays a basic item-item autoencoder architecture. As seen in the left side of the figure, the encoder takes in the high dimensional input vector and encodes it to a lower dimensional space (which is shown in the green), while the decoder then takes the lower dimensional vector and outputs a vector with the same original dimensions. The intuition behind the autoencoder's outstanding performance is that the architecture extracts important features from the input and results in a representation that is more accurate than the input. Our architecture consists of an item-item autoencoder with one hidden layer. The output of the autoencoder is expressed with different values than the input, although it has the same dimensions.

2.3 Implicit vs Explicit Feedback Datasets

This section is about the difference between implicit and explicit datasets and explains their benefits and downsides.

In the past, many recommender systems had different rating systems that were used to represent a user's interest in a single item. Such rating systems required interactions from users, primarily from surveys or questionnaires that asked to rate a certain item on a scale. A simple example could consist of asking a user to rate a movie from 1 star to 5 stars after the user had finished watching it. It was difficult to gather data on past user and item interactions due to the nature of surveys but recently, there has been more research that has been conducted on using implicit feedback from the user [7, 11–14], a method that makes gathering data much easier.

Implicit feedback datasets usually do not consist of data that is obtained by explicitly asking for a user's thoughts or ratings on a certain item. Instead, implicit data is usually collected from some action such as a user's click or purchase of a particular item. Although researchers do not obtain specific information on the extent of how much a user likes a certain item, they ultimately receive data that a user indeed did express some sort interest in an item. Implicit feedback datasets consist only of 0's and 1's, where a 0 represents no interest in a certain item, while a 1 represents some sort of expressed interest. Due to the nature of these binary datasets, collecting data and creating new datasets becomes much easier and less intrusive to the users' experiences. Implicit feedback provides much less information about the extent of how much a user likes a particular item compared to explicit feedback, where data can be expressed on a measurable scale, but has become more popular due to the ease in being able to gather data.



Explicit Feedback

	i_0	i_1	i_2	i_3	i_4
u_1	0	1	3	0	5
u_2	0	0	0	1	2
u_3	2	1	0	0	0
u_4	4	2	1	0	0

Implicit Feedback

	i_0	i_1	i_2	i_3	i_4
u_1	0	1	1	0	1
u_2	0	0	0	1	1
u_3	1	1	0	0	0
u_4	1	1	1	0	0

Figure 2.2: Explicit and implicit feedback example vectors.

Figure 2.2 displays the difference between a dataset that utilizes explicit feedback versus a dataset that utilizes implicit feedback. As we see on the left, each of the users have an expressed rating of certain items from a scale of 1-5, while on the right, the same users and items are expressed with 0s and 1s. As mentioned before, explicit feedback datasets provide more details on the strength of a user's preference towards a certain item and implicit datasets tend to be inherently noisy due to the lack of such data.

2.4 Related Works

There has been research conducted on graph-based autoencoders and curator-based recommendation, but research on a graph convolutional based autoencoder model has not been explored yet.

Autoencoders Meet Collaborative Filtering (AutoRec): Autoencoders have proven to be effective in several others domains, with one of them being recommender systems [6]. Past research has shown that autoencoder frameworks combined with collaborative filtering for user-based autoencoding and item-based autoencoding have state of the art performances, especially when compared to other recommender system baselines. While the model we propose uses also autoencoders to produce item embeddings for a given user, our research also factors in user-curator interactions as well as a graph convolutional computational step to differentiate it from the AutoRec model.

Graph Convolutional Matrix Completion (GCMC): Graphs are able to represent and model user-item interactions, so many graph-based models for recommendation exist. Graph learning algorithms have shown the potential to learn complex relationships between each of the nodes in the graph. Existing research has been conducted in representing user-item interactions via a graph and using an autoencoder to reconstruct the user-item graph [15]. The creators of the graph convolution matrix completion (GCMC) model used a bilinear interaction between the user and items to feed through the encoder and decoder. Unlike the model we propose, the GCMC research seeks to reconstruct the user-item graph itself and does not incorporate user-curator interactions with an implicit feedback dataset.

User Recommendation in Content Curation Platforms (CuRe): Content curation platforms allow for a variety of different recommendations to be made: users can be recommended both and items and other curators. The creators of the CuRe model provide an implementation with the primary purpose of recommending users other curators as well as a supplementary task of recommending users other items [7]. The CuRe model uses separate denoising autoencoders for curators and items and combines both of those representations in an attention layer to provide recommendations for both items and curators. The model we propose focuses primarily on a top-k recommendation of items to users, while relying on the interactions of user and curators through a graph structure.

3. THE PROPOSED MODEL

In this section we propose a novel graph convolutional autoencoder recommender system with user-item and user-curator interactions to provide top-k item recommendations. Alluding to our previous YouTube example, items are examples of videos, users are examples of people who consume the videos, and curators are examples of channels that can create videos and also channels that users can subscribe to. Section 3.1 will cover the problem formulation and specific notations, while Section 3.2 will go in-depth about our proposed solution.

3.1 Problem Formulation

Our goal of top-k item recommendations was to output the top-k items from the set of all items in the dataset for each user.

Let $U = \{u_1, u_2, ..., u_N\}$ be the set of the users in the dataset, where N is the total number of users. From this point on, we will call the people a single user follows *curators*.

For a user u, we use a binary vector $c_u = [c_{u1}, c_{u2}, ..., c_{uN}]$ to represent the users that are followed by u. In our case, a 0 would represent the user u not following a curator, while a 1 would represent the user u following a curator.

Let $I = \{i_1, i_2, ..., i_M\}$ be the set of the items in the dataset, where M is the total number of items.

For a user u, we use a binary vector $y_u = [y_{u1}, y_{u2}, ..., y_{uM}]$ to represent the items that user u is interested in. Likewise, a 0 would represent the user u being uninterested in an item, while a 1 would represent the user u being interested in an item.

3.2 Proposed Solution

Our intuition is that we can harness the interactions between users and curators to provide more accurate recommendations. A graph model is the most logical structure to use because it can clearly represent all of the users and the curators each user follows in the same space. In our graph structure, we represent each user as a node and connect each user-curator interaction with an edge. Each user, with its own user-item interactions, can be represented by an item embedding previously defined in Section 3.1 as y_u. As a result, we end up encapsulating user-item and user-curator interactions by building a single graph.

With this graph structure, we propose to harness the curators each user follows to construct an item embedding that more accurately represents each users' preferences. A graph convolution calculation allows us to both utilize the information from connected curators and reconstruct a given user's item embedding to potentially provide more information about their preferences.

The first section entails the similarity calculation that is performed between each user and curator that is followed by that user. It then describes the user-curator graph convolution operation that occurs as a preprocessing step before detailing the autoencoder architecture in the second section.

3.2.1 User-Curator Graph Convolution

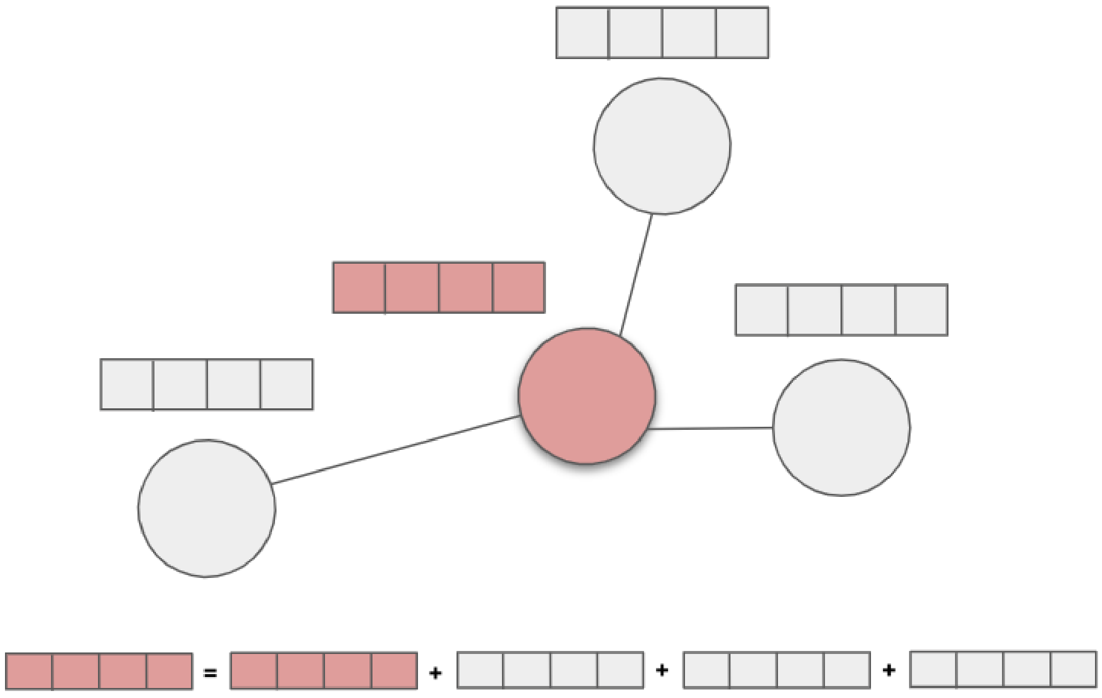


Figure 3.1: Graph convolution example.

Figure 3.1 displays a basic example of one graph convolution operation for a user. Given the one-hop neighbors of the original node, the graph convolution operations occurs by aggregating the neighbors into the original vector to form a new embedding. Before the convolution operation, the similarity between the one hop neighbors and the original node must be calculated with

$$S_{i,j} = \frac{1}{P}\sum_{p=1}^{P}(y_{ip})(y_{jp}) \tag{Eq. 3.1}$$

Due to the complex nature of user-curator relationships and preferences, the similarity calculation is done to reduce the noise that is received from the data. In this case, P is the total number of items that the original user and the specific curator are interested in. Essentially, the similarity calculation finds the proportion of items a user j shares in common with the original user i. A single similarity computation between a user and a curator results in a constant ranging from 0 to 1, where a 0 represents a relationship with absolutely no similarities and a 1 represents a relationship with identical preferences.

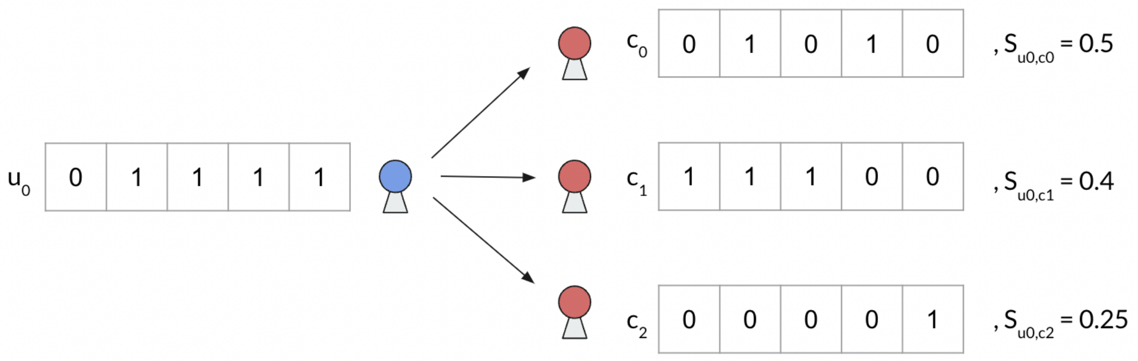
Now that we have defined the similarity calculation, we can express one iteration of the graph convolution operation for a user as the following

$$y_u = y_u + \sum_{c=1}^{C} S_{c,u}(y_c) \tag{Eq. 3.2}$$

The convolution calculation redefines the item embedding for a user u. The similarity between u and each curator that is followed by u is calculated and is then used to scale the curators' item vectors. We can observe that each similarity value limits the impact that each curator has on the redefined item embedding for u.

From the extra data obtained from the convolution operation and the filtering of noise from the similarity calculation, we modified the dataset to include more information about the users' preferences. After the convolution operation, the embedding for the user u would equal the aggregation of all of the curators' item vectors, scaled by each similarity value in addition to the original user-item interactions.

3.2.2 *Graph Convolution Preprocessing Step Example*



u_0 0 1 1 1 1
c_0 0 1 0 1 0 , $S_{u0,c0} = 0.5$
c_1 1 1 1 0 0 , $S_{u0,c1} = 0.4$
c_2 0 0 0 0 1 , $S_{u0,c2} = 0.25$

Figure 3.2: Graph convolution computation.

To give an example computation, the next few figures show what the graph convolution preprocessing step would look like if it were simplified to a small number of users and items. Figure 3.2 displays the user-item interactions with the vector next to the users and displays the user-curator interactions with arrows going from the user to the curators. We can observe that user u_0 follows three curators, c_0, c_1, c_2, in this example.

Given the data, the first calculation that we walk through is the similarity calculation. In order to evaluate how similar a user and a curator are, we use Eq. 3.1 to calculate the ratio of the number of items a given user and a curator share together. The first similarity computation between user u_0 and curator c_0 can be done by observing that there are a total of 4 items that u_0 and c_0 interact with. Of those four items, we see that there are only two items they share in common. As a result, we take those two numbers as a ratio and come out with a result of $S_{u_0,c_0} = \frac{2}{4} = 0.5$. The remaining similarities for c_1 and c_2 can be computed by following the same computation method.

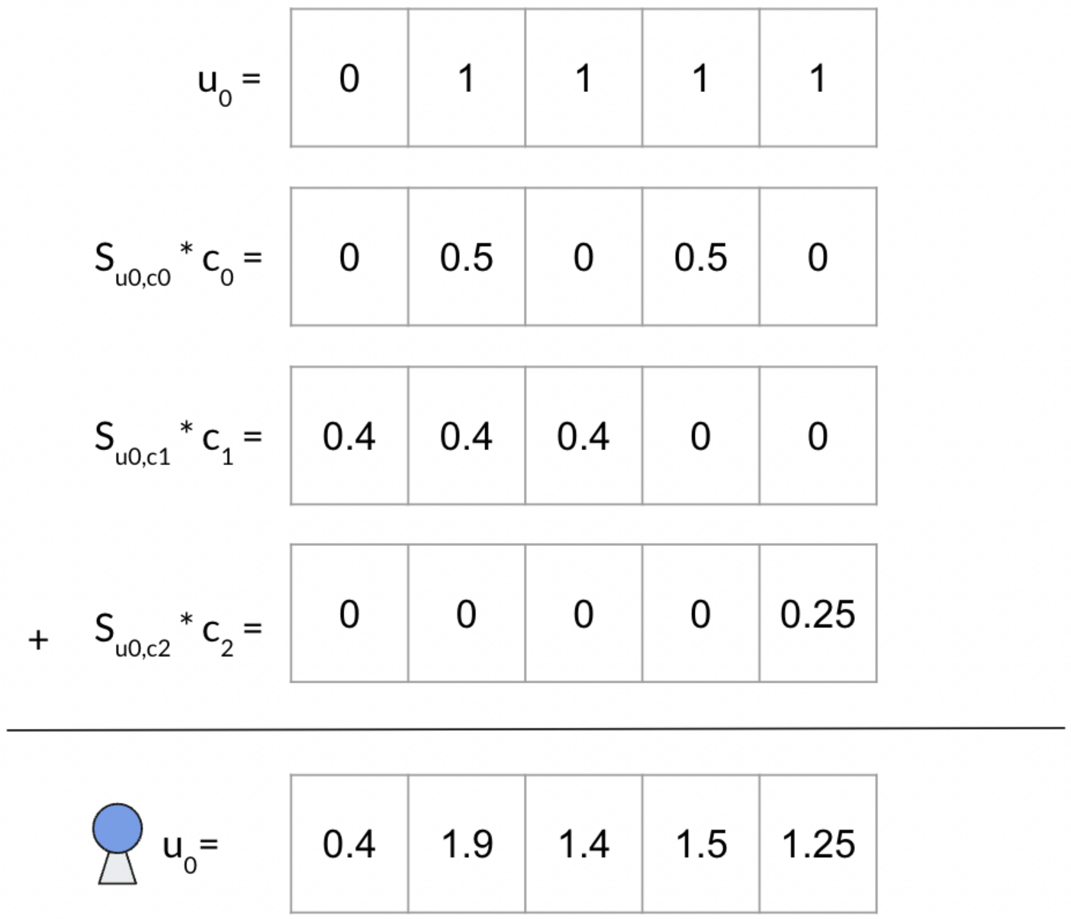


Figure 3.3: Modifying original user vector embedding.

With the similarity calculation finished, the next step is to do the graph convolution computation. Figure 3.3 displays a graphic on how the graph convolution computation is completed for a single user. Given the similarity values for each user and curator pair, we multiply the similarity constants by the original curator vectors. After doing so, we add up all of the curator vectors and add it to the original user embedding, essentially redefining the user vector to contain more information about the curators they follow.

3.2.3 Autoencoder Architecture

After preprocessing graph convolutional steps for all users in the dataset, the data is fed into a basic autoencoder with one hidden layer.

We can feed a binary vector $y_u = [y_{u1}, y_{u2}, ..., y_{uM}]$, with the graph convolutional preprocessing step completed for the data, as an input to the autoencoder architecture represented as

$$h_u = \delta(Vy_u + b) \tag{Eq. 3.3}$$

where V is the weight vector for the encoder, $\delta(\cdot)$ is the sigmoid function, and b is the bias term. The binary vector y_u in this case represents the user-item interaction vector for a single user. The encoder transforms the input into an embedding that is expressed in a lower dimension latent space, which is a compressed space that consists of specific features taken from the original input.

The decoder takes the output of the hidden layer, h_u, and performs a calculation that transforms h_u to the same dimension as original input represented by the equation

$$\hat{y}_u = \delta(Wh_u + b') \tag{Eq. 3.4}$$

where W is the weight vector for the decoder, h_u is the output of the encoder and b' is the bias term. y_u is the resulting output vector is the re-expressed vector of the original input vector and will be used to generate item predictions for the user.

3.2.4 Loss

Since our model only consists of an autoencoder architecture, the loss is just the error calculated from the reconstruction of the original vector

$$L = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y_n})^2 \tag{Eq. 3.5}$$

The ultimate goal of using the loss function is to reduce the amount of error in our predictions, and so we used the mean squared error for our loss function, where N is the total number of items, y_n is the original item embedding and $\hat{y_n}$ is the predicted vector. A lower loss implies that our model has greater chance of predicting the users' preferences.

3.3 Additional Modifications to the Model

We further explore several modifications to the graph convolutional autoencoder model, such as including creating pseudo-curators and experimenting with floor and ceiling thresholds.

3.3.1 Pseudo-curators

We hypothesize that adding more curators would lead to a greater amount of data that we could utilize to redefine each user embedding, so we create pseudo-curators. These curators are ones that we create from the existing data and are not originally included in the dataset.

For a given user u_0 and its user-item interaction vector, y_{u0} we take all of the items that u_0 has interacted with and randomly sample β amount of them. From those items, we compile all of the curators that also interacted with the randomly sampled items and count the total number of times each curator has interacted with the randomly sampled items. We then take the top 3 curators that have interacted with the items the most and made the curators "pseudo-curators". The pseudo-curators are added to the user-curators interactions so that the graph convolutional preprocessing step applies to both curators and the pseudo-curators.

3.3.2 Thresholds

With the graph convolutional operation, we hypothesize that there might be too much noise being added into the new user embeddings, so we propose to add ceiling and floor thresholds.

Because each user in the dataset follows a different number of curators, we set ceiling and floor thresholds to be relative to the number of curators that they follow. Because the convolution operation increases the overall values of the user embeddings the more curators a user followed, we implement a ceiling value of 1. If certain user-item embeddings are over a certain value relative to the number of curators a user followed, we can set those values to the ceiling value of 1. Likewise, for the floor thresholds, if certain user-item embeddings are under a certain value, those embeddings are set to the value value of 0.

4. EXPERIMENTS

4.1 Metrics

In order to evaluate model performance, we use precision and recall as the metrics. In order to explain precision and recall metric evaluations for our recommendation system, we can express certain terminologies with the following examples: if our model predicts that a user will like a particular item and the user ends up actually liking it, we get a true positive; if our model predicts that a user will like a particular item and the user does not end up liking it, we have a false positive; if our model predicts that a user will not like a particular item but the user ends up liking it, we have a false negative; if our model predicts that a user will not like a particular item and the user ends up not liking it, we have a true negative.

We can more specifically define precision with Equation 4.1 and recall with Equation 4.2.

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{Eq. 4.1}$$

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{Eq. 4.2}$$

As we can see both of the metrics use the true positive as a ratio of different elements. Precision emphasizes the proportion of true positives that we received out of all the items we thought a user would like. In contrast to that, recall emphasizes the proportion of true positives that we received in proportion to all of the items that a user actually liked. These precision and recall metrics were calculated and averaged for all of the users in the dataset.

4.2 Baselines

We use an implicit feedback collaborative filtering model and an item-item autoencoder as baselines to compare our graph convolutional autoencoder model results:

- Collaborative Filtering: There are several different types of collaborative filtering models, but the one we use as a baseline is an implicit feedback collaborative filtering model that uses an alternating least squares optimizer. Essentially, collaborative filtering models take past user-item interactions and recommends items to a user, based on other users who like similar items. The baseline that we use is a separate optimized version, specifically for implicit feedback datasets, where they used confidence scale to indicate that a user was more likely to prefer a certain item.

- Item-item Autoencoder: The other baseline that we use is a simple item-item autoencoder with a one single hidden layer. We make the encoder have an input that is the same dimension as the total number of items and make the decoder have the same output as the total number of items.

4.3 Results

Because we want to recommend a specific number of items that users are likely to consume, we use a top-K method of recommending items to users. The method in which top-K metrics are generated is by confining the number of predictions we make to be equal to K. For example, if we generate 10 recommendations for a user and only 5 are true positives, our precision at K=10 would be 0.5. Likewise, we would repeat the calculations, generating our top-1, top-5, ..., top-K predictions for each precision and recall metric. To evaluate the top-K recommendations, we use the precision and recall at k=1, 5, 10, 15, and 50 metrics for each user, where the metrics are evaluated at the number of items that are predicted correctly for each user.

We generate precision and recall metrics for two different versions of our model: One with a similarity calculation during the convolution calculation and one without a similarity calculation during the convolution calculation. We want to show the differences between the results with and without a crucial element for denoising the data. Our model with the similarity convolution calculation is labeled as GCA+, while the model without the similarity convolution calculation is labeled as GCA-. As for the model hyperparameters, we trained the model using batch gradient descent, using a batch size of 256 over a total of 50 epochs.

The dataset that was used is very sparse, which helps explain the overall lower performance of the recommender system algorithms that were trained. We use a real life Spotify dataset, with songs acting as the items and curators as people who created music playlists for users to consume. In total, there are approximately 9,000 users and 70,000 items in our dataset.

We see in Table 4.1 that the collaborative filtering algorithm performed generally the best, with our GCA+ model barely performing better for the top K=1 recommendations. We see that our proposed graph convolutional autoencoder model was generally similar in performance compared to the AutoRec model.

Table 4.1: Comparing the baselines with the proposed model for the top-K item recommendation of items where K=1,5,10,15,50, using precision and recall as the evaluation metrics

Spotify Dataset										
Model	Precision (%)					Recall (%)				
	K=1	K=5	K=10	K=15	K=50	K=1	K=5	K=10	K=15	K=50
Collaborative Filtering	2.80	3.26	3.08	2.95	2.34	0.065	0.384	0.718	1.96	2.71
AutoRec	3.09	2.26	1.94	1.80	1.41	0.069	0.256	0.439	0.611	1.60
GCA-	2.50	1.95	1.78	1.66	1.25	0.057	0.222	0.406	0.565	1.42
GCA+	3.13	2.27	1.94	1.82	1.40	0.071	0.257	0.441	0.621	1.59

One thing to also note are the differences in performances between the GCA algorithm with and without the similarity calculation. We see again in Table 4.1 that without the similarity calculation, the precision and recall performances are worse for each top-K recommendation compared to GCA with the similarity calculation.

Also seen in Table 4.1 the differences in performance between the machine learning based algorithms compared to the collaborative filtering algorithm. We see that in the top K=1 predictions, the collaborative filtering algorithm has a worse precision and recall performance, outperforming the other models at the K=5,10,15,50 levels.

Table 4.2: Precision and recall of pseudo-curators and thresholds for K=1,5,10,15,50

Model	Precision (%)					Recall (%)				
	K=1	K=5	K=10	K=15	K=50	K=1	K=5	K=10	K=15	K=50
GCA+ with Thresholds	3.08	2.27	1.99	1.82	1.41	0.069	0.257	0.450	0.621	1.61
GCA+ with Pseudo-curators	3.10	2.24	1.98	1.82	1.40	0.070	0.255	0.450	0.621	1.60
GCA+ with Pseudo-curators and Thresholds	0.42	0.82	1.15	1.13	0.79	0	0	0.002	0.004	.009

4.4 Pseudo-Curator and Threshold Results

The results of the pseudo-curator and threshold implementations are shown in Table 4.2 The precision and recall values are similar in value compared to the GCA+ model and Autoencoder baseline shown in Table 4.1 There are marginal increases in some of the metrics, but no noticeable improvements even with pseudo-curators and threshold implementations. One thing to note is that the model performance with pseudo-curators and thresholds combined dropped considerably, especially when compared to other implementations.

4.5 Findings

As we can see in the table, we had marginal improvements with our graph convolutional autoencoder model for top K=1 recommendations. We also saw a significant difference in precision and recall metrics with and without the similarity calculations for all top-K values. Since implicit feedback datasets are known to be inherently noisy, our results reflect the importance of the similarity calculation. It is difficult to deduce whether the similarity calculation has a direct impact on the performance on the model, but we can definitely see that the GCA+ model converges more quickly compared to the GCA- model.

We used the graph preprocessing step to try and capture more patterns of interaction between users and curators, but it surprisingly did not yield results that were signficantly greater than the baseline models. Instead, we were met with results that were very similar to the autoencoder model, which is expected as we in fact did use the autoencoder architecture after the preprocessing step. From these results, it can be seen that our graph preprocessing step did not have a significant enough impact to improve the recommendations given to users.

The same can be said about the implementations of the pseudo-curators and ceiling and floor thresholds. We expected to see more distinct results with the model modifications of our pseudo-curators and thresholds but were met with similar results to the basic autoencoder architecture. We hypothesized that the pseudo-curators would lead to more accurate predictions due to the augmentation of the users' original data but failed to find a great deviation from the baseline autoencoder model. With the ceiling and thresholds, we had also hypothesized that normalizing the dataset to be more representative of an implicit feedback dataset would provide better results but were also met with similar results.

One observation that can be made is that while our model had worse performances compared to the authors of [7], who used the same dataset, we used only one-third of the total 27k user-item interaction data, which can explain the lower model performance. In the dataset that we used, there were only roughly 9k user-curator interactions, which why we chose to not use the rest of the dataset. Deep learning usually requires a large dataset to obtain sufficient results so using more users for training could result in more definite results.

5. CONCLUSION

In this thesis, we proposed a graph-based model to more accurately provide top-K item recommendations to users in a content-curator based platform. We were able to produce a model that had performances similar to that of the state of the art autoencoder and marginally higher precision and recall performances for the top k=1 recommendations, while the our model fell short of the collaborative filtering baseline model.

By comparing the model performances with and without the similarity computations, we showed that just incorporating the user-curator data as the training data resulted in additional noise that decreased model performance. The relationship between the graph-based connections of users and curators is still a concept that has great potential for future research in representing the interconnectedness of users and curators in a content-curator platform.

In addition to the graph convolution computations, we created pseudo-curators and applied thresholds to control the distribution of the input data. Although, the results were similar to that of the baseline autoencoder, the concept of data augmentation via pseudo-curators is still a topic to be further researched. Furthermore, testing our model with other curator datasets may offer different insights.

REFERENCES

[1] R. Burke, “Hybrid recommender systems: Survey and experiments,” *User Modeling and User-Adapted Interaction*, vol. 12, pp. 331–370, 2004.

[2] J. A. Konstan and J. Riedl, “Recommender systems: from algorithms to user experience,” *User Modeling and User-Adapted Interaction*, vol. 22, pp. 101–123, 2011.

[3] R. Burke, “Hybrid web recommender systems,” in *The Adaptive Web*, 2007.

[4] D. Goldberg, D. A. Nichols, B. M. Oki, and D. B. Terry, “Using collaborative filtering to weave an information tapestry,” *Commun. ACM*, vol. 35, pp. 61–70, 1992.

[5] M. J. Pazzani and D. Billsus, “Content-based recommendation systems,” in *The Adaptive Web*, 2007.

[6] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, “Autorec: Autoencoders meet collaborative filtering,” *Proceedings of the 24th International Conference on World Wide Web*, 2015.

[7] J. Wang, Z. Zhu, and J. Caverlee, “User recommendation in content curation platforms,” *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020.

[8] Y. He, J. Wang, W. Niu, and J. Caverlee, “A hierarchical self-attentive model for recommending user-generated item lists,” *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[9] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester, “Collaborative denoising auto-encoders for top-n recommender systems,” *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, 2016.

[10] Z. Zhu, J. Wang, and J. Caverlee, “Improving top-k recommendation via jointcollaborative autoencoders,” *The World Wide Web Conference*, 2019.

[11] J. Wang, K. Ding, Z. Zhu, and J. Caverlee, “Session-based recommendation with hypergraph attention networks,” *ArXiv*, vol. abs/2112.14266, 2021.

[12] Y. Hu, Y. Koren, and C. Volinsky, “Collaborative filtering for implicit feedback datasets,” *2008 Eighth IEEE International Conference on Data Mining*, pp. 263–272, 2008.

[13] S. Kabbur, X. Ning, and G. Karypis, “Fism: factored item similarity models for top-n recommender systems,” *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

[14] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, “Bpr: Bayesian personalized ranking from implicit feedback,” *ArXiv*, vol. abs/1205.2618, 2009.

[15] R. van den Berg, T. Kipf, and M. Welling, “Graph convolutional matrix completion,” *ArXiv*, vol. abs/1706.02263, 2017.