Exploring Graph Neural Networks in E-Commerce Recommendation Systems

University of Wisconsin, Stevens Point
M.S. in Data Science
Bryce Fenlon
December 2022

Acknowledgements

To my wife, Fanni, for her unbelievable patience over the last two years, and to my fellow students and professors who have shaped my journey through the Data Science program. I wish you all the best.

Abstract

With the rise of big data, recommendation systems have evolved from simple information retrieval and filtering to complex neural networks recommending products from billions of data points. While collaborative filtering methods have been the predominant method for recommendation, regardless of whether it is implemented classically or with neural networks, different data structures are now allowing for a wider variety of recommendation models. Graph data structures are able to retain critical, non-Euclidean relationship-level data, allowing us to train models that have a better implicit understanding of the underlying data. This project is a case study comparing graph-centric models to more traditional recommendation systems. Benchmarking results show promise for graph neural network ability to recommend products more accurately in small to medium-sized E-commerce applications, with minimal tradeoffs in model development (training time, computational overhead, etc.).

Contents

List of Tables

List of Figures

Chapter 1: Introduction

Background

Over the last four decades, E-commerce has grown from small online marketplaces to a multitrillion-dollar behemoth that basically defines global retail markets. It permeates nearly every aspect of our lives, and it is increasingly driven not by human decisions but by algorithmic ones in the form of recommender systems. Recommender systems have evolved over time to deliver targeted recommendations based on a myriad of factors, from previous purchases and similar user profiles to more complex, session-based information, cookies stored on local systems, and even social media knowledge-graphs.

A large majority of recommender systems (RS) can be classified as content-based, collaborative, or a hybrid of both (Tang et al, 2021). Content-based RS rely on item representations – turning abstract content such as descriptions into algebraic representations – and user profiles that represent preferences. Product descriptions, keywords, and characteristics of items that the user previously purchased are used to filter additional products to offer the best recommendation.

Collaborative filtering (CF), on the other hand, assumes that if user **A** and user **B** both buy the same item, user **A** is more likely than a random individual to buy a different item that user **B** also bought. CF systems can be item-based or user-based, but the basic concept revolves around matrix factorization. The majority of modern recommender systems are based on some form of CF, though it is not without downsides.

Collaborative filtering systems often suffer from the “cold-start” problem, where new users have little to no information about their preferences and therefore recommendation

becomes intractable. In the same vein, new items have not been rated and there is simply not enough information to provide a ranking. Cold-start difficulties also stem from the structure of bipartite graphs themselves, as users are only connected to items (and vice-versa), meaning that there are no direct relationships between similar users or similar items.

Sparsity is another problem, as CF relies on ratings and the percentage of users who rate items is quite low. CF, like content-based recommendation, relies on the assumption that the data being used to train and validate the model is Euclidean in nature. A different paradigm is needed when approaching data that cannot be properly embedded in Euclidean space.

Research Motivation

In recent years, graph-based recommendation systems have garnered a great deal of interest (Wang et al., 2021b). This stems from the observation that most data in recommender systems fit neatly into a graph structure either explicitly or implicitly. For example, we see that user-item relationships are explicit, while shared user preferences and shared characteristics of items can be thought of as implicit relationships. Graph data structures are incredibly data-rich, with relational information encoded in the edges and the structure of features implicitly defined by the geometry of the graph itself. There is no sensible way to abstract this data into higher dimensions of Euclidean space.

Graph Neural Networks (GNNs) seek to process this non-Euclidean graph data directly, so a distinct advantage of leveraging GNNs in recommendation systems is that we circumvent the task of embedding our graph-structured data in Euclidean space. Instead of matrix factorization, graph learning and GNNs for recommendation solve a link prediction problem. A

key advantage of graph-based recommendation systems over CF is that the "cold-start" problem can be at least partially overcome by leveraging "multi-hop relationships" (Wu et al., 2022), which allows item or user data to diffuse or propagate through other user or item nodes, since no direct links exist on either set of a bipartite graph.

The first motivation for this project is to explore recommendation systems and their evolution in E-commerce, with a focus on graph methods. The second motivation is to explore current state-of-the-art implementations for graph-based recommendation systems. The third motivation is to benchmark more typical recommendation systems, such as collaborative filtering and traditional deep learning models, against graph-based models. The final motivation is to cement graph recommendation methods for use in small to medium-sized E-commerce

Scope of Topic and Objective

Directed by the motivations outlined above, this project will explore Graph Neural Networks and their applications in recommendation systems. GNNs "[have] achieved remarkable success in recommender systems in the past few years" and "GNN-based models outperform previous methods and achieve new state-of-the-art results" (Wu et al., 2022, p. 2). The scope of this project will be:

a) Review recommendation systems' evolution
b) Review GNNs and why they are suited to recommendation tasks
c) Exploration of 1 GNN model in recommendation
d) Conclusion, highlighting tradeoffs between directly capturing non-Euclidean nature of graph data and ease of implementation

Limitations

While GNNs for recommendation have seen a veritable explosion in interest recently, the body of research surrounding them is still quite young compared to more established deep learning frameworks (i.e., CNNs and RNNs). As a result, definitions between articles and studies are not necessarily consistent and the field is still coalescing.

Another significant obstacle will be processing power. While the end result of training a GNN model can perhaps outperform other methods, training GNNs requires significant processing overhead. I plan to implement my GNN models using Google Colab GPUs, in order to stay within a Jupyter Notebook-like platform while leveraging the large increase in computing power that GPUs provide for neural network training.

Organization of the Paper

I. Introduction
    a. Background
    b. Research motivation
    c. Scope and objective
    d. Limitations
II. Literature Review
    a. Evolution of recommendation systems
    b. Current state of the art

c. Summary of existing body of research surrounding graph neural networks in recommendation

d. Review three promising approaches to recommendation

    i. Collaborative filtering

    ii. Neural collaborative filtering

    iii. Graph learning

III. Methodologies

a. Review of data

b. Model implementations and architectures

c. Benchmarking

IV. Results

a. Benchmarking results

V. Conclusion

a. Results summary

b. Implications for GNN implementation in E-commerce and conclusion

Chapter 2: Literature Review

Recommender Systems

Prior to the advent of the internet and modern information systems, recommender systems (RS) were called something else entirely: word-of-mouth (Kembellec & Chartron, 2014). Community and individual relationships, socioeconomic status, and the qualities of a product might have informed recommendations for a product to a given person. This would be considered a social system because the motivations for recommendation in this paradigm originate purely from individual experience. These systems have been around for millennia. Thousands of years ago, word-of-mouth recommendations determined where religions spread and what farmers cultivated. During later periods of history, ministers and courts recommended to kings and queens which lands to invade and what laws to make (Sharma & Singh, 2016). Things continued on like this for some time, and while the industrial age, technology, and the internet allowed for different "word-of-mouth" methods (e.g., newspaper and television ads), there was still no targeting. With the advent of the internet, though, product recommendation took on a whole new meaning.

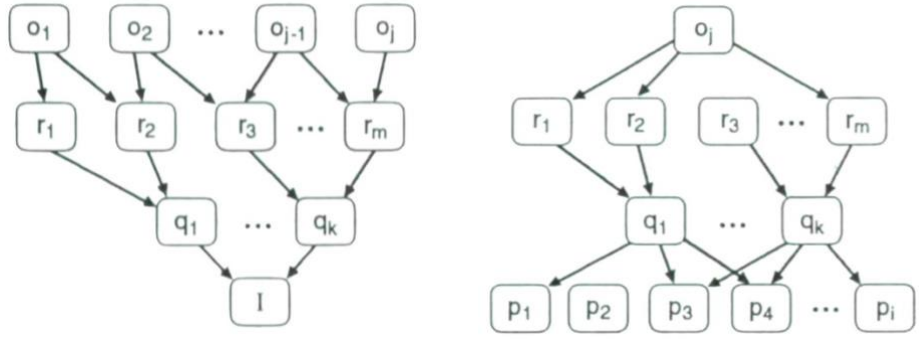
Word-of-mouth recommendation never really disappeared, though, and it even developed new aspects. Today, people will rely on word-of-mouth recommendations in online product reviews. When considering the confluence of word-of-mouth recommendations and the internet, however, trustworthiness comes to the foreground. Previously, when consumers actually knew the person making the word-of-mouth recommendation, trust was implicit. Online, though, trust is a trickier thing (Lee & Hong, 2019).

As the internet grew into being in the 1980s, so too did online marketplaces and what would come to be known as electronic commerce (E-commerce) platforms. E-commerce platforms have exploded in popularity since then, especially since the turn of the millennium (Oestreicher-Singer & Sundararajan, 2012), and product recommendation strategies have evolved right along with them. But before their application to E-commerce and product recommendation, RS developed out of necessity.

Information overload was the spark that turned RS into a distinct area of research (Kembellec & Chartron, 2014). The concept of information overload has remained a central topic in RS literature. The topic diverged, though, into two distinct areas. One simply being how to overcome information overload (i.e. recommendation methods themselves), and the other investigating the relationship between information load, perceived overload, RS recommendation sets, and how user attitudes and behaviors toward recommendations change in response to these variables (Aljukhadar et al., 2012; Wang et al., 2021a). The conversation around information overload has matured in concert with RS.

Belkin and Croft (1992) examined information retrieval and information filtering and found that the two are very similar on an abstract level, using the example of Bayesian inference networks to show that retrieval and filtering are "two sides of the same coin" (p. 37). This can be very clearly seen in Figure 1, and the general implication is that RS closely relates to the process of information filtering. From this, we can deduce that recommender systems are specializations of information filtering, and that information filtering is a specialization of information retrieval; information retrieval → information filtering → recommender systems.

Figure 1



Bayesian inference networks for information retrieval (left) and information filtering (right). From Information filtering and information retrieval: two sides of the same coin? *Communications of The ACM, 35*(12), p. 34 by Belkin, N.J. and Croft, W.B., 1992.

As RS grew, it was soon applied to a wide variety of applications in the E-commerce industry. One of the clearest early examples of RS employed in a product recommendation setting used decision trees to select personalized advertisements for customers (Kim et al., 2001). This marked a clear shift from a geographic market share perspective to "earning the market share of an individual consumer" (Kim et al., 2001, p. 45).

It would not be an area of research without dissenting opinions, however. Over the last two decades there have been many studies focusing on automated recommendation's effect on E-commerce. Some focus on the fact that RS often recommends similar products or, in a similar tone, that RS should bring more diversity to recommendations (Bodoff & Ho, 2015; Nguyen & Hsu, 2022; Park & Han, 2013). Others have pointed out that the data which RS models are trained and deployed on is biased, affecting the very foundations of RS (Tsekouras, 2017). Granted, none of these researchers were advocating for getting rid of RS completely, but rather for careful consideration of all the factors before deployment.

Collaborative & Content-based Filtering

More than just implementations of recommendation, collaborative filtering and content-based filtering are the twin pillars upon which nearly all other systems are based. It makes sense, therefore, to first look at how they developed.

Information overload, while setting RS apart as a field of study, also spurred the development of the first collaborative filtering program (Dong et al., 2022). Tapestry was developed by the Xerox Palo Alto Research Center in 1992 for purposes of filtering relevant electronic documents (Goldberg et al., 1992). Tapestry relied on explicit annotations of relevance or implicit feedback – such as replying to an E-mail – alongside an indexed document store to filter messages to an individual user's "little box" (Goldberg et al., 1992, p. 63). Up until this point, interest in RS was almost entirely academic or focused on productivity. With the introduction of GroupLens, however, Paul Resnick et al. (1994) showed the promising commercial applications of RS by applying collaborative filtering to news recommendation. John Riedl, one of the initial developers of GroupLens, took his knowledge and applied it to the E-Commerce space with his company Net Perceptions, which developed online shopping solutions for companies like Amazon, 3M, and Kmart (University of Minnesota, 2013). Riedl also continued to advance research in RS by starting the aptly named GroupLens Research lab at the University of Minnesota.

The earliest recommender systems, like GroupLens, were collaborative and used nearest-neighbor methods in giving predicted ratings. The advantage of nearest-neighbor collaborative filtering is that it can be considered a "lazy learning approach" (Schafer et al., 2001, p. 118). This means that the model can be built and updated in real time while the RS

gives recommendations. Other methods of collaborative filtering, like Bayesian networks, require an offline model-building period before deployment. For environments where consumer preferences do not need to be updated regularly, Bayesian networks can provide results similar to nearest-neighbor methods with models that are smaller and faster (Breese et al., 1998; Schafer et al., 2001). Prior to 2005, collaborative filtering was the dominating force in RS studies and practical applications.

Problems with collaborative filtering, though, soon became apparent in both their measures of performance and their limitations in application (Kembellec & Chartron, 2014). Collaborative filtering systems focused primarily on conventional accuracy metrics such as Pearson's Correlation Coefficient (PCC) and Cosine similarity (Ai et al., 2022; Dong et al., 2022). These measures are statistically sound but misrepresent the true performance of collaborative filtering. In deployment, ratings are sparse and new items and users are constantly being added to the model. One early effort to at least partially overcome the cold start and sparsity problems used inferred popularity of the item within the dataset to "prime" the recommendation engine (Ahn, 2006). However, this method can backfire and create a sort of echo chamber, where only popular products tend to be favored (Kembellec & Chartron, 2014). Later, collaborative filtering was combined with genetic algorithms in an attempt to overcome both the sparsity and scalability problems by more efficiently optimizing and condensing the user-item matrix (Kim & Ahn, 2011).

It was not until 2006 that the discussion about RS performance measurement started to center around user experiences rather than just accuracy metrics (Dong et al., 2022; McNee et

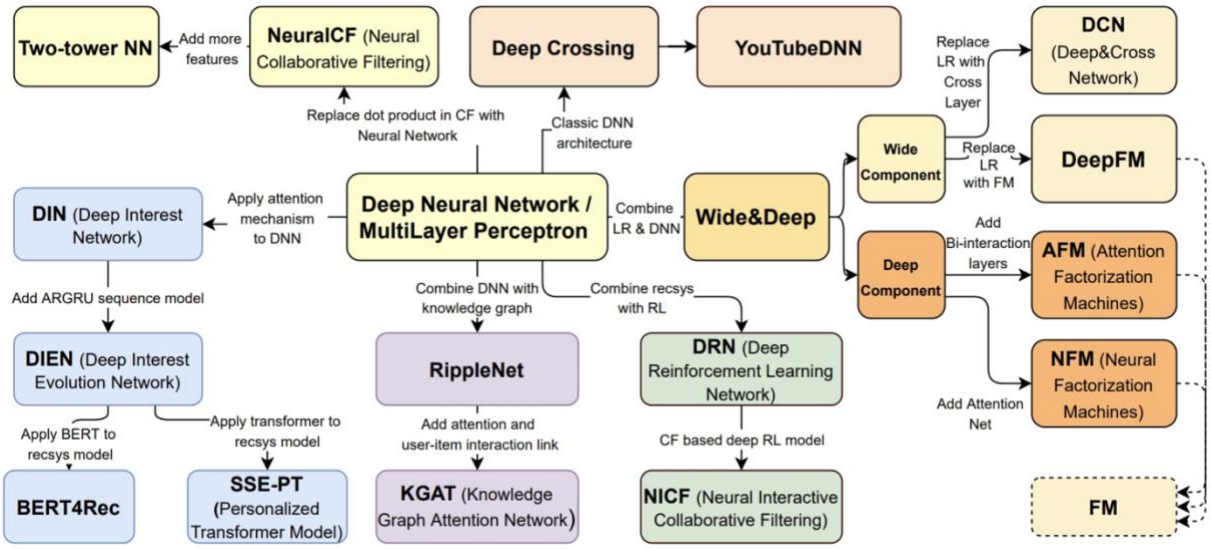
al., 2006). Though collaborative filtering saw advances after 2005, the limitations of collaborative methods in web-scale deployment settings are difficult to overcome.

Content-based filtering is another paradigm upon which most current RS are based. At its core, content-based recommendation is a logical solution to the problem of recommendation. Looking at content that a consumer has previously bought, it would only be a small logical leap to assume that they would be interested in content that is similar to what they have previously showed interest in. Of course, this leads to overspecialization in content-based filtering, where recommendations are *very* similar to products or subjects that the consumer has already bought. For example, a user might enjoy mostly Sci-Fi books, but with no recommendations outside of Sci-Fi, they may not have a good experience with the recommendation engine if they want to explore outside of their typical genre.

Current State of the Art

Most current state of the art recommendation algorithms are classified as deep learning algorithms (Dong et al., 2022). Figure 2 shows a clear picture of current state of the art deep learning methods in recommendation. They all stem from the MultiLayer Perceptron (MLP) – or Deep Neural Network (DNN) – model, where there are many hidden layers between the input and final output.

Figure 2



Taxonomy of deep learning recommendation models. From A Brief History of Recommender Systems. *arXiv preprint,* by Dong, Z., Wang, Z., Xu, J., Tang, R., and Wen, J., 2022

Within this taxonomy, there are several main branches. Neural Collaborative Filtering (NeuralCF) takes the concept of matrix factorization – a collaborative filtering method popularized by the Netflix competition – and uses MLP instead of simple matrix multiplication to "enhance feature crossing capacity" (Dong et al., 2022, p. 3). Wide and deep models are another school of thought, combining two models: a traditional logistic regression model with strong memory (*wide*) and a DNN that can generalize better (*deep*) by embedding dense representations of users and items in lower-dimensional space (Dong et al., 2022). The last main branch in this taxonomy (other than graph-centric models, which will be discussed in the next section) includes models based on attention mechanisms or transformers, which introduce NLP concepts and technologies to improve recommendation.

Even more recently, researchers are attempting to move beyond correlation in recommendation – the main motivation behind collaborative filtering and many deep learning

applications – and toward a causal inference model of recommendation (Dong et al., 2022). Causal inference frameworks are currently being developed to overcome significant and unavoidable consequences of correlation-based recommendation models in use today. These downsides include data bias, missing data or data noise, and the inability of current RS to achieve objectives of "fairness, explainability, [and] transparency" (Gao et al., 2022, p. 3). Causal inference aims to give fairer, more transparent, and explainable recommendations by embracing an entirely different paradigm of recommendation. However, causality and causal inference are outside the scope of this paper and will not be discussed further.

Graph Neural Networks in Recommendation

As recommendation methods have evolved over the years, they have not done so in a static environment. Data structures and data storage methods have been in a near constant state of flux over the last three decades, owing to the burgeoning amount of data on the internet and the rise of "big data". Geometric deep learning – Graph Neural Networks (GNNs) specifically – arose from the confluence of graph data development, advances in recursive neural networks, and geometric concepts of invariance (Bianchini et al., 2005). While the supporting work for aspects of geometric deep learning and GNNs have been around for decades now (Zhou et al., 2021), it is only recently, in the last 10 to 15 years, that graphs have begun to play a bigger role in deep learning and consequently RS (Bianchini et al., 2005; Weng et al., 2011; Wu et al., 2022). Most data in RS can be represented by graph structures, with users and items interacting only with the other class in what is called a bi-partite graph, so it is a logical step to approach the problem from the perspective of graphs.

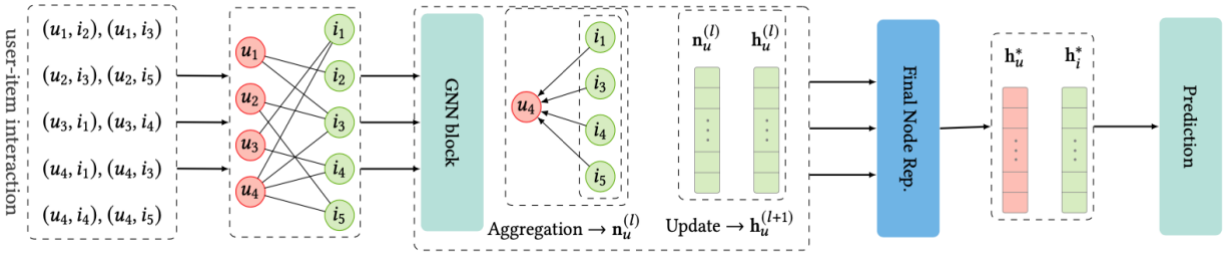
Ignoring for a second the recommendation aspect, graphs in neural network applications became of interest around the same time as RS solidified as a field of study. Sperduti and Starita (1997) proposed a generalized form of a standard neuron, the "complex recursive neuron" (p. 715), which allowed them to dynamically build topological representations of complex structures using recursive neural networks. Topology and invariance in output with respect to input is an important concept to geometric deep learning and GNNs (Bianchini et al., 2005; Lahtinen et al., 2000). GNNs can explicitly encode graph topology, which benefits classification and regression tasks by improving representation learning – the method of embedding user and item data in a shared vector space (Scarselli et al., 2013; Wu et al., 2022). Regarding invariance, rearranging nodes in a graph while preserving the edges might represent the same set of relationships – or flow of information in a directed graph – but in reality, it corresponds to a different graph.

Early efforts to apply neural networks to graph data either extracted features invariant to transformation or preprocessed the graph data (Lahtinen et al., 2000; Nordström et al., 2000). Approaching the invariance problem this way allowed Recursive Neural Networks, and later Recurrent Neural Networks and Feedforward Neural Networks, to handle graph data under specific circumstances in certain applications. They were limited, though, by their core functionality being defined by state transition systems (the transition from node v_i to v_j where nodes represent states of the system) which could not accurately represent the underlying data in deployment and made it difficult to expand the methodology to larger graphs – web-scale recommenders for example (Zhou et al., 2020). With the development of Convolutional Neural

Networks applied to Euclidean data, it was not long before it was applied to the graph domain and recommendation (Yu et al., 2022).

When it comes to recommendation in general, there are a variety of sub-tasks that can be completed. Some of these include sequential recommendation, social recommendation, knowledge graph-based recommendation, points-of-interest recommendation, group recommendation, bundle recommendation, click-through-rate prediction, and multimedia recommendation (Wu et al., 2022). However, here we will consider only user-item collaborative filtering implemented in a GNN structure. The general structure of this task can be seen in Figure 3.

Figure 3



The overall framework of GNN in user-item collaborative filtering. From Graph Neural Networks in Recommender Systems: A Survey. *ACM Comput. Surv., 37*(4), p. 9 by Wu, S., Sun, F., Zhang, W., Xie, X., & Cui, B., 2022.

Focus of This Project

For this project, we will focus exclusively on three models for recommendation. As an absolute baseline for recommendation, we will use a relatively simple matrix factorization approach (Koren et al., 2009). From an industry perspective this is quite antiquated, but it will provide a good metric for the other models.

For a more traditional deep learning implementation we will use a Neural Collaborative Filtering (NeuralCF) model. Collaborative filtering has often been the go-to method for

recommendation in deployment. While traditional implementations such as matrix factorization models have efficiency issues with large numbers of items and users, NeuralCF has been shown to have constant time complexity through the use of approximate nearest neighbor search (Dong et al., 2022). This NeuralCF model will serve as our baseline for deep learning recommendation.

Lastly, the GNN model we will benchmark against the other two models will be a Graph Convolutional Network called LightGCN. This model was developed by He et al. (2020). LightGCN builds off of an intermediate model structure called Neural Graph Collaborative Filtering and removes the extraneous functionality that is unnecessary when considering only user-item interactions (He et al., 2020).

Chapter 3: Methodology

Overview of Goal

As discussed in chapter 2, Graph Neural Networks are unique among deep learning algorithms because they are able to explicitly encode non-Euclidean data in their structure. The goal of this project is to determine their efficacy in E-commerce on a small to medium scale, as more complex recommender systems – such as GNNs – are typically implemented across larger, more complex ecosystems (i.e., Pinterest or Amazon).

Collaborative Filtering Model

Collaborative filtering (CF) can be classified as either neighborhood-based or model-based, with matrix factorization falling into the latter (Yuan et al., 2019). Matrix factorization (MF) recommenders map both users and items to a latent factor space, where "user-item interactions are modeled as inner products in that space", visualized in Figure 4, such that user-item ratings are estimated as:

Figure 4

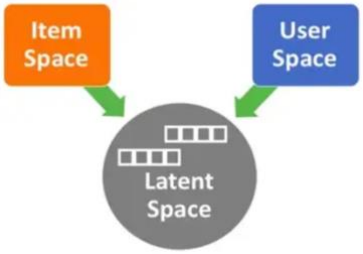


Illustration of joint latent factor space in matrix. Retrieved from https://medium.com/

$$\hat{r}_{ui} = q_i^T p_u \quad (1)$$

Where $q_i^T p_u$ is the dot product between user u and item i latent vectors (Koren et al., 2009, p. 44). For this project, a particular version of MF will be used: singular value decomposition (SVD).

SVD is a technique that was first developed for information retrieval systems but was prohibitively expensive because SVD encounters difficulties with missing values – a problem that is all too common in sparse recommendation data – and required data imputation (Koren et al., 2009). It was not until later that methods were developed to model the observed ratings only, training the data only on observed user-item ratings and applying a regularization parameter (Koren et al., 2009). In this way, SVD models follow Equation 2, where r_{ui} is the known rating, λ is the regularization parameter determined through cross-validation or hyperparameter optimization, and κ is the set of user-item pairs that we have known ratings for.

$$\min_{q*,p*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \quad (2)$$

SVD was popularized for recommendation by an independent software developer Simon Funk in the Netflix prize competition. In his own words, Simon describes SVD like this:

> The best way to understand SVD is probably in reverse: to look at how one reconstructs a data matrix from the singular vectors. Consider just a single column-vector A and corresponding row-vector B. If you multiply A by B you get a matrix as tall as A and as wide as B. Now if you have some target data matrix of the same size (say the Netflix movie-by-user ratings matrix) you can ask: What choice of A and B would make that reconstructed matrix as close to my target matrix as possible? SVD is a mathematical trick for finding that optimal AB pair. (Piatetsky, 2007, p. 39)

Funk also popularized stochastic gradient descent (SGD) for minimizing Equation 2. This is a fairly straightforward procedure where the prediction error e_{ui} (Equation 3) modifies the parameters q_i (Equation 4) and p_u (Equation 5) in the opposite direction of the gradient of Equation 1. In Equations 4 and 5 γ can be described as a momentum term, which dampens harmonic oscillations of the SGD algorithm and helps to accelerate convergence while preventing premature convergence in a local minimum (Qian, 1999). Counterintuitively, even though SGD updates the parameters during each training loop, it is more efficient because it does not have to compute the gradient of the objective function across the entire training set, as is the case in “normal” gradient descent.

$$e_{ui} \stackrel{\text{def}}{=} r_{ui} - q_i^T p_u \tag{3}$$

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i) \tag{4}$$

$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u) \tag{5}$$

Implementation of the SVD collaborative filtering model comes from a previous class, DS775: Prescriptive Analytics. It is built, tuned, and cross-validated in Python using the surprise package (Hug, 2020).

Neural Collaborative Filtering Model

Neural collaborative filtering (NeuralCF) can be thought of as a fusion of two different model approaches – generalized matrix factorization (GMF) and multilayer perceptron (MLP) – combining the strengths of each in modeling the latent factor space (He et al., 2017).

GMF is exactly what it sounds like, a generalization of MF which is represented by Equation 6 (He et al., 2017, eq. 9),

$$\hat{r}_{ui} = a_{out}\left(\mathrm{h}^T(\mathrm{p}_u \odot \mathrm{q}_i)\right) \tag{6}$$

$$L = -\sum_{(u,i)\in Y \cup Y^-} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}) \tag{7}$$

where a_{out} is the activation function and h^T is a matrix of edge weights, learned through SGD optimization of the log loss, represented by Equation 7 (He et al., 2017, eq. 7). The user latent vector is represented by $\mathrm{p}_u = P^T v_u^U$ and the item latent vector by $\mathrm{q}_i = Q^T \mathrm{v}_i^I$, with v_u^U and v_i^I representing the user and item feature vectors, respectively. Additionally, $P \in \mathbb{R}^{M \times K}$ and $Q \in \mathbb{R}^{N \times K}$ represent the latent factor matrices for users and items, respectively, with *M* number of

users, *N* number of items, and *K* representing the dimension of the latent space. The ⊙ symbol represents the element-wise product of vectors p_u and q_i, important for finding the minimum of multivariable functions using the Hessian (Qian, 1999). When a_{out} is an identity function (i.e., when the input equals the output) and when $\mathbf{h}$ is a uniform vector of 1, the output layer of GMF represented by Equation 6 becomes vanilla MF.

The MLP path in NeuralCF gives the model more flexibility and embodies nonlinearities which GMF cannot (He et al., 2017). Equation 8 defines the MLP model according to He et al. (2017, eq. 10).

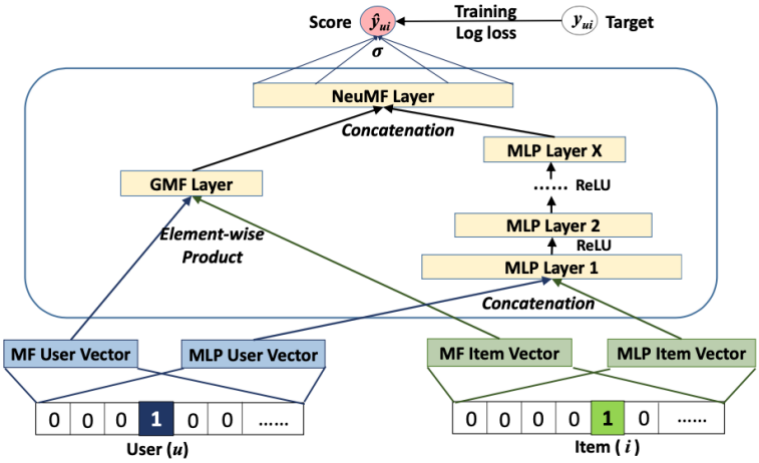
$$\begin{aligned} \mathrm{z}_1 &= \phi_1(\mathrm{p_u}, \mathrm{q_i}) = \begin{bmatrix} \mathrm{p_u} \\ \mathrm{q_i} \end{bmatrix}, \\ \phi_2(\mathrm{z}_1) &= a_2(\mathrm{W}_2^T \mathrm{z}_1 + \mathrm{b}_2), \\ &\ldots\ldots \\ \phi_L(\mathrm{z}_{L-1}) &= a_L(\mathrm{W}_L^T \mathrm{z}_{L-1} + \mathrm{b}_L) \\ \hat{r}_{ui} &= \sigma\left(\mathrm{h}^T \phi_L(\mathrm{z}_{L-1})\right) \end{aligned} \tag{8}$$

W_L here defines the weight matrix, b_L the bias, and a_L the activation function for a given layer L in the MLP model. $\phi_L(\mathrm{z}_{L-1})$ can be thought of as a condensed notation for all the layers of the MLP model, which would then be expanded to the following (He et al., 2017, eq. 12):

$$\phi_{\mathrm{L}}(\mathrm{z}_{L-1}) = a_L\left(\mathrm{W}_L^T\left(a_{L-1}\left(\ldots a_2\left(\mathrm{W}_2^T\begin{bmatrix} \mathrm{p}_u \\ \mathrm{q}_i \end{bmatrix} + \mathrm{b}_2\right)\ldots\right)\right) + \mathrm{b}_L\right) \tag{9}$$

At its most basic, implementing NeuralCF simply adds another layer in the neural network which is then initialized with the pre-trained GMF and MLP models. The input to the NeuralCF layer is the concatenation of the GMF and MLP output ("Concatenation" in Figure 5).

Figure 5



General implementation of NeuralCF. From Neural Collaborative Filtering. *Proceedings of the 26th International Conference on World Wide Web, WWW '17,* p. 177 by He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.S., 2017.

Taken together, then, the final output of the NeuralCF model developed by He et al. is represented by Equation 10 (2017, eq. 12).

$$\hat{r}_{ui} = \sigma\left(\mathrm{h}^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix}\right) \qquad (10)$$

Where $\phi^{GMF} = \mathrm{p}_u^G \odot \mathrm{q}_i^G$ and $\phi^{MLP} = a_L\left(\mathrm{W}_L^T\left(a_{L-1}\left(\ldots a_2\left(\mathrm{W}_2^T \begin{bmatrix} \mathrm{p}_u^M \\ \mathrm{q}_i^M \end{bmatrix} + \mathrm{b}_2\right)\ldots\right)\right) + \mathrm{b}_L\right)$ (He et al., 2017, eq. 12). Here, the G and M superscripts simply refer to GMF and MLP embeddings, respectively, for both users and items. This is a necessary distinction because the embeddings are not forced to be the same size in NeuralCF along the two model paths, giving some more flexibility to the model and allowing NeuralCF to "[combine] the linearity of MF and non-linearity of DNNs for modelling user-item latent structures" (He et al., 2017, p. 177).

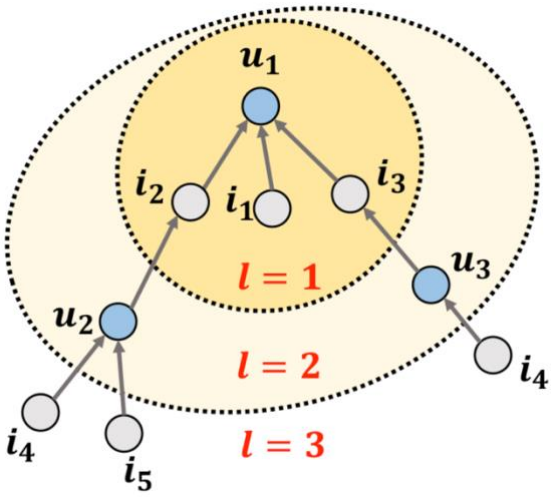
NeuralCF will be implemented in Python following the framework developed by He et al. (2017). For this model, Keras and Tensorflow will be used to build the MLP and GMF layers. In the MLP, the output activations for the layers will a rectified linear unit (ReLU) which works

better with the sparse data that is typically found in recommender systems (He et al., 2017). Code for implementing NeuralCF comes from He et al. (2017) and was adapted for use in Jupyter Notebooks. Optimization of the network will be done with the Adam optimizer

GNN Model

When comparing GNNs to MF or NeuralCF, one of the biggest differences is that instead of defining user and item embeddings and a latent factor space where interactions between users and items are represented as inner products, GNNs can model the relationship directly by preserving the interaction between users and items in the embedding layer (He et al., 2020). This preserves the non-Euclidean nature of the data, as previously mentioned.

Figure 6



User-item connectivity and propagation. From Neural Graph Collaborative Filtering. *Proceedings of the* 42nd *International ACM SIGIR Conference on Research and Development in Information Retrieval*, *SIGIR* '19, p. 166 by Wang, X., He, X., Wang, M., Feng, F., & Chua, T.S., 2020.

As a halfway point between pure GNN and NeuralCF, Neural Graph Collaborative Filtering (NGCF) was introduced in 2020 by Wang et al. While this project focuses on GNN, or to be specific Graph Convolution Network (GCN), NGCF and GCN have some elements in common. The first is the concept of message passing or propagation, visually represented in Figure 6, where neighbor embeddings (user-item embeddings) are represented by Equations 9 and 10 as described by He et al. (2020, eq. 3).

$$e_u^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} e_i^{(k)} \quad (9)$$

$$e_i^{(k+1)} = \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} e_u^{(k)} \quad (10)$$

Where $e_i^{(k)}$ and $e_u^{(k)}$ represent item and user embeddings that have been propagated k layers, which then informs the neighbor embedding for the node at $k+1$. $\frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}}$ takes the places of Lasso (L1 normalization) or Ridge (L2 normalization) regression to prevent overfitting (Kipf & Welling, 2017; He et al., 2020). $\mathcal{N}_u$ represents the set of items that user u interacts with, while $\mathcal{N}_i$ represents the set of users that item i interacts with. In the implementation of GCN that will be explored for this project – LightGCN – the only trainable parameters are at the target nodes of the graph ($e_u^{(0)}$ and $e_i^{(0)}$), with higher layers computed using Equations 9 and 10 above. Finally, the weighted sums for user and item embeddings are computed following:

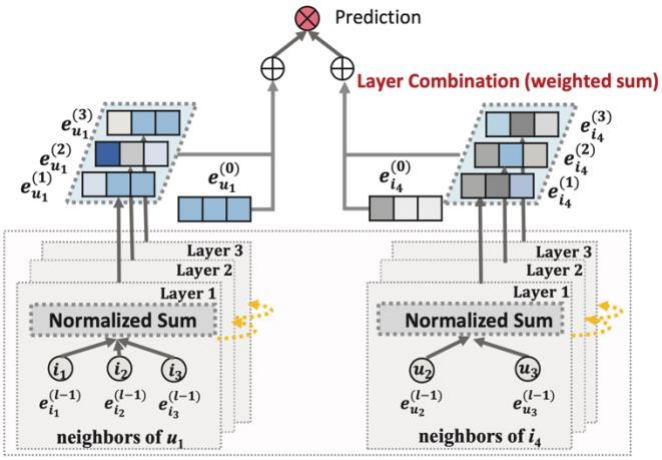
$$e_u = \sum_{k=0}^{K} \alpha_k e_u^{(k)};\ e_i = \sum_{k=0}^{K} \alpha_k e_i^{(k)} \quad (11)$$

where α_k here is defined as $1/(K+1)$ in order not to overcomplicate the model (He et al., 2020, eq. 4). Once the final weighted sum is given for both user and item, the model prediction is given simply as their inner product. Equation 12 (He et al., 2020, eq. 5) is almost identical to Equation 1 because while the math that derives e_u and e_i may be different, the implementation of the result is basically the same: we predict the user-item score based on the inner product of learned vectors, whether those vectors come from a shared latent vector space or from GCN embeddings of higher-order interactions.

$$\hat{y}_{ui} = e_u^T e_i \quad (12)$$

Visually, the GCN model from He et al. (2020) can be seen in Figure 7 – termed LightGCN – with the normalized sum neighbor embeddings represented by Equations 9 and 10, and the weighted sum of the layers represented by Equation(s) 11.

Figure 7



Depiction of LightGCN model. From LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '20*, p. 642 by *He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M., 2020.*

When it comes to training the GCN model, the authors minimize the Bayesian Personalized Ranking loss, which is able to capture the relationship between a user and two distinct items (He et al., 2020; Rendle et al., 2009).

$$L_{BPR} = -\sum_{u=1}^{M}\sum_{i\in\mathcal{N}_u}\sum_{j\notin\mathcal{N}_u}\ln\sigma(\hat{y}_{ui}-\hat{y}_{uj}) + \lambda\left\|\mathrm{E}^{(0)}\right\|^2 \qquad (13)$$

Where λ is the L2 regularization parameter (Ridge regression) and $\ln\sigma(\hat{y}_{ui}-\hat{y}_{uj}) = \frac{1}{1+e^{-\hat{y}_{ui}+\hat{y}_{uj}}}$, which is essentially the probability that the user prefers item i to item j (Rendle et al., 2009, p. 455). Equation 13 (He et al., 2020, eq. 15) shows BPR loss as originally proposed by Rendle et al. in 2009, adjusted for the specifics of LightGCN. Where BPR loss normally regularizes all model parameters with λ, LightGCN only has to train the 0-th layer embeddings so in this case $\Theta = \mathrm{E}^{(0)}$ (Rendle et al., 2009, p. 455).

LightGCN is implemented in Python using a prebuilt package named `recommenders`, provided by Miguel Fierro, a Data Scientist and Software Engineer at Microsoft. Code examples are supplied through GitHub, and the `recommenders` package is available through PyPI.

Benchmarking

With regards to evaluating the models' performance, two metrics will be used. Hit ratio and root-mean-square error (RMSE). RMSE is a very common statistical measure of fit between predicted values and ground-truth values in a model:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2}$$

where $Y_i - \hat{Y}_i$ is the difference between true and predicted ratings for a given item (or in this case, movie) and n is the number of items.

Hit ratio (HR) is an even simpler measure, though it can vary depending on the data and model being used. For example, Alsini & Huynh propose:

$$\mathrm{HR} = \frac{m}{\min\left(n_R, n_G\right)}$$

where n_G is the number of ground truth values and n_R is the number of recommendations given from the model (Alsini & Huynh, 2020, p. 2). This works quite well for a given user on social media that might use multiple hashtags or only a single hashtag but does not fit with movie recommendations where the model predicts a single value for a given user-item interaction. In our case, we will define the hit ratio as:

$$\mathrm{HR}_K = \frac{R_{hit}^K}{R_{all}}$$

where K is the number of recommendations given, R_{hit}^K is the count of user recommendations for which ground truth values for validation are in the recommendation list of length K, and R_{all} is the count of all users that recommendations were generated for. For our purposes we will test hit ratio at 20, so we will find the ratio of users for which holdout values are present in

the top 20 recommendations. If a user has multiple hits, we will simply count the user as a positive example (i.e., contributing 1 to R_{hit}^{K}).

Testing methodology will follow a typical train-test split (85-15 split). The train and test data will be the same for all 3 model implementations, using a seed value of 14 for sklearn.model_selection.train_test_split. RMSE and HR_K will both be calculated on the test set. By combining these two measures, we will gain a better understanding of whether the models are creating high quality ratings in general (RMSE) and if the recommendations are relevant (HR).

Description of Data

For this project, two MovieLens datasets from GroupLens Research are used. The MovieLens 100K Dataset and the MovieLens 1M Dataset are both used (Harper & Konstan, 2015), in order to provide a more realistic gauge for small to medium sized business use cases. While not technically an E-commerce dataset like Amazon datasets, MovieLens datasets provide something that most other open source datasets do not – a reliable, industry standard benchmark within the field of recommendation. A brief overview of both datasets is seen below in Table 1. While both datasets include users that have only rated at least 20 movies each, it should be noted that these are not true 20-cores; that is, we do not assume that each movie is also rated by 20 unique users.

Table 1

Description of MovieLens datasets

	MovieLens 100K	**MovieLens 1M**
# Ratings	100,000	1,000,209
# Movies	1,682	3,706
# Users	943	6,040

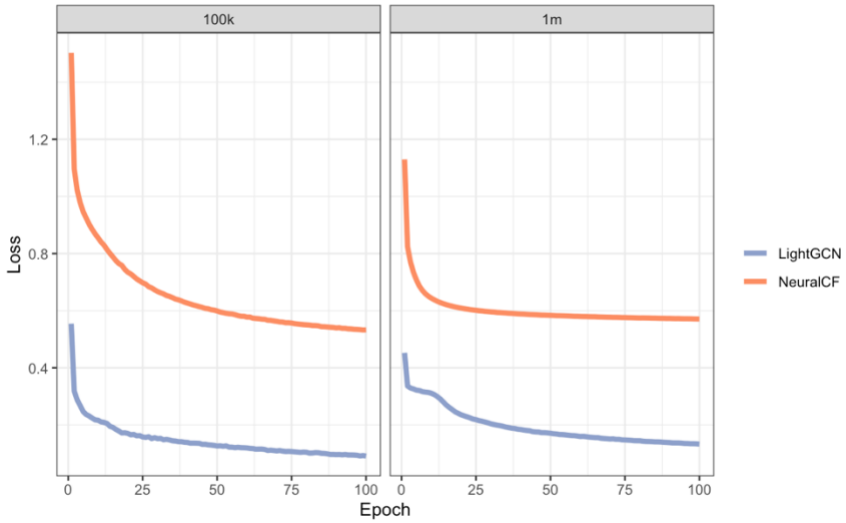
Hyperparameter Tuning & Training

For hyperparameter tuning, GridSearchCV on the SVD from the surprise package in Python will be used on the 100k and 1M datasets separately. The suggested hyperparameters for learning rate, decay (regularization), and training duration (epochs) will be used across all 3 models for the training stage. For the NeuralCF model, we utilize the Adam optimizer which controls two separate regularization parameters, so only the learning rate is set (Kingma & Ba, 2014). For tuning, we optimized the hyperparameters on the SVD model with 85% of the data for both datasets. Learning rate will be set at 0.0025 for the 100k models, and 0.0005 for the 1M models. Regularization – or decay of the learning rate – is set to 0.1 in the CF model for the 100k dataset and 0.01 in the CF model for the 1M dataset, and finally 10^{-4} in the LightGCN model. Decay will remain unchanged from the Keras Adam optimizer defaults in the NeuralCF model. All models will train for 100 epochs, because GPU compute time is expensive and finite.

Chapter 4: Results

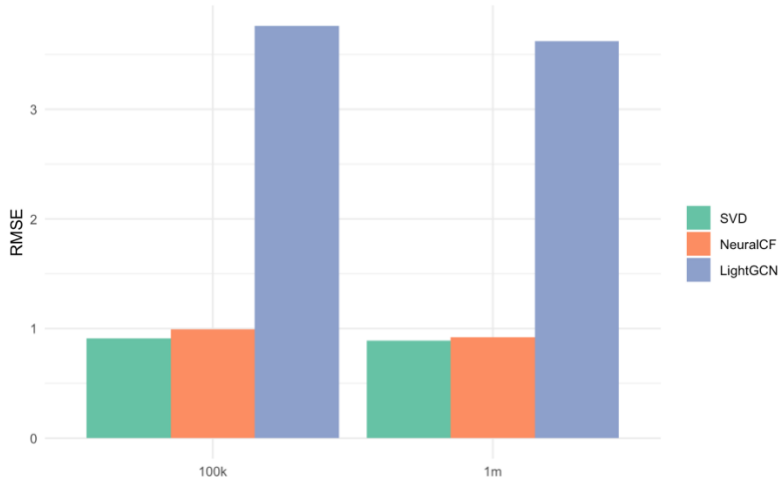
Benchmarking Results

Figure 8



During the training of the models, some very clear trends emerged. First, the training loss curves (Figure 8) for NeuralCF and the LightGCN implementation show very clearly the impact that the problem has on loss function optimization. This project focuses exclusively on recommendation, so it makes sense for the loss equation to incorporate ranking and recommendation. However, the NeuralCF model trained with Keras in Python uses only the mean absolute error of the predictions while LightGCN calculates BPR loss and compares items to one another.

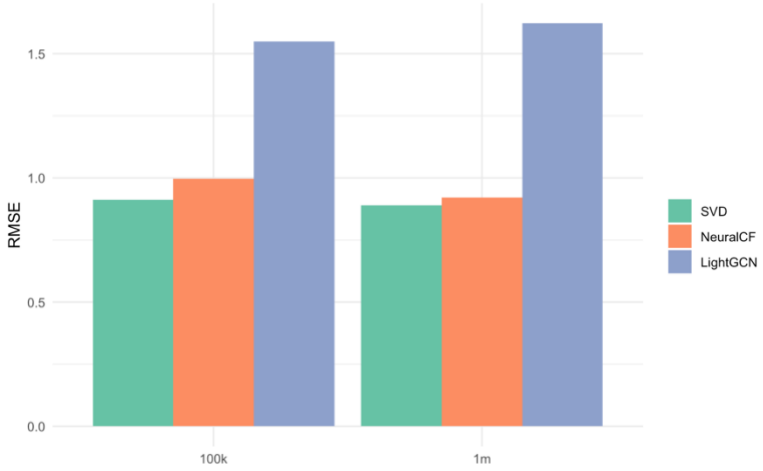
Figure 9



While the train loss is interesting, it does not fall within the scope of this benchmarking. RMSE for the 3 models (Figure 9) is quite far off from what we might expect – more complex models tend to give more accurate predictions and therefore a better, or lower, RMSE. However, both the NeuralCF and LightGCN models color outside the lines when predicting scores between users and items. In the training process for these models the output is not

explicitly forced to be rangebound. Rather, the model weights and embeddings are learned through training and output is on a continuous scale. Because of this, outputs for the NeuralCF and LightGCN models can far exceed the 1-5 ranking range.

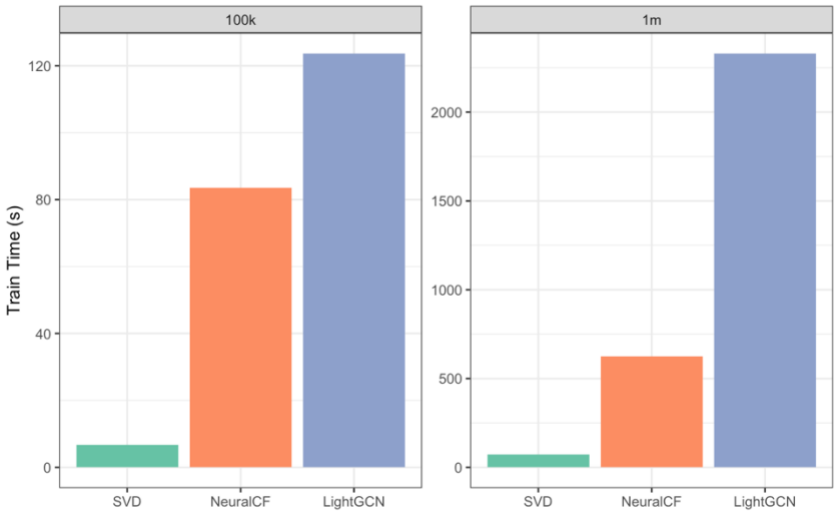
Figure 10



In the case of LightGCN, we can slightly compensate by normalizing the result with sklearn's MinMaxScaler function and the result is Figure 10. When normalizing the RMSE we also find that the model trained for the 1M dataset performs worse based on RMSE. This is likely to do with the fact that for larger datasets LightGCN outputs a wider range of values and normalizing that wider range distorts the direct comparison between dataset sizes. RMSE only tells part of the story though.

Another key factor to note is the training time, visible in Figure 11. For smaller data, like the 100k Movielens dataset, LightGCN training time is only about 50% longer than NeuralCF, while for larger datasets LightGCN training time starts to grow much faster

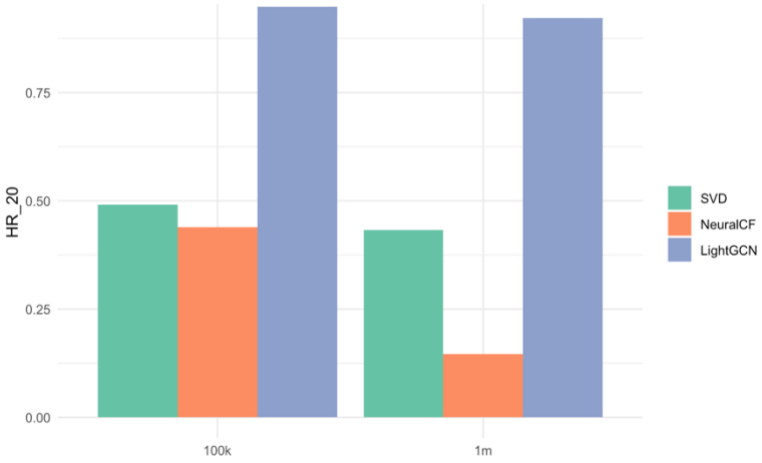
Figure 11



than NeuralCF (more than 250% longer for the 1M Movielens dataset). To put this difference in training time into perspective, we also need to look at our calculated Hit Ratio (HR_{20}).

Again, HR_{20} by our definition is the ratio of users for which holdout recommendation items appear in the top 20 recommendations by the model to the total number of users in the test set. For LightGCN, 92.2% of users in the 1M test set were recommended movies that they had actually rated in the holdout data (94.8% in the 100k test set). In comparison, SVD was less than 50% for both sets of data and NeuralCF even less than that.

Figure 12



Chapter 5: Discussion and Conclusion

Summary of Results

Taken individually, these results could make a convincing case for the SVD model. Training time is lower, and performance is better than NeuralCF on both dataset sizes considering both RMSE and HR_{20}, and it is a popular, proven model that is regularly used in recommendation systems. However, HR_{20} results for LightGCN cannot be ignored. Compared to SVD, more than twice as many users had top 20 recommendations that were present in the holdout data for testing. For “small” data, like our 100k dataset, training time was approximately 20x longer for the LightGCN model compared to SVD but still only about 2

minutes in total. Overall, LightGCN presents higher quality recommendations for users with a minimal tradeoff in training time for small to medium-sized datasets that are typically seen in smaller E-commerce systems.

Limitations and Recommendations

As an extension of this work, more time and energy should be given to a direct comparison of NeuralCF and LightGCN. We see a pronounced decrease in HR_{20} for NeuralCF between the 100k and 1M models. This is likely an issue with tuning the decay – or regularization rates – for the Adam optimizer, inadequate model size, training length, or perhaps a combination of these. Since all these models were trained only for 100 epochs, we have a better comparison model to model, but individually their performance could be improved with more in-depth hyperparameter tuning. For example, for the 1M dataset, it might make more sense to increase the latent factor space for NeuralCF above 10 to better account for increased variability in the training data over the 100k dataset. In this project as many parameters as possible remained the same between models and datasets, in order to facilitate the benchmarking. This may have been to the detriment of model performance, however, and bears further investigation to build a fuller picture of how LightGCN and NeuralCF compare.

A facet of recommendation was also left out of this investigation, namely the 'cold start' problem in collaborative filtering. In every case, the train-test split was stratified by user. This means that no user in the test data was *not* present in the training data. Of course, this does not reflect reality. For LightGCN this was a requirement of the model implementation, as the testing phase did not allow for users not in the training data. Necessarily, this had to be extended to the other models. Because the scope of this investigation was centered purely on

model performance and not performance in production, the ‘cold start’ part of the problem was ignored. There is a great deal of literature on overcoming this problem, however. LightGCN and any graph model may be particularly amenable to the use of knowledge graphs of users, user metadata, or social network data.

Conclusion

The results shown above clearly show the efficacy of graph methods for recommendation, with LightGCN offering better recommendation relevance to a set of users over SVD and NeuralCF. In this project, we:

a) Discussed the evolution of recommendation systems and their core characteristics, as well as reviewing graph neural networks and their role in recommendation systems
b) Explored and implemented several recommendation systems
c) Benchmarked the results of recommendation on data relevant to the E-commerce industry

For recommendation systems based purely on user-item interaction data, graph methods are a definite improvement over more traditional collaborative filtering or neural network models. The final takeaway is that models leveraging graph data structures can be a better option in deployment for small to medium-sized E-commerce providers, with only a minimal tradeoff in training time.

Sources

Ahn, H.J. (2006). Utilizing Popularity Characteristics for Product Recommendation. *International Journal of Electronic Commerce, 11*(2), 59-80. https://doi.org/10.2753/JEC1086-4415110203

Ai, J., Cai, Y., Su, Z., Zhang, K., Pend, D., & Chen, Q. (2022). Predicting user-item links in recommender systems based on similarity-network resource allocation. *Chaos, Solitons, and Fractals, 158*. https://doi.org/10.1016/j.chaos.2022.112032

Aljukhadar, M., Senecal, S., & Daoust, C.E. (2012). Using Recommendation Agents to Cope with Information Overload. *International Journal of Electronic Commerce, 17*(2), 41-70. https://doi.org/10.2753/JEC1086-4415170202

Alsini, A. & Huynh, D.Q. (2020). Hit Ratio: An Evaluation Metric for Hashtag Recommendation. *arXiv preprint*. https://doi.org/10.48550/arXiv.2010.01258

Basu, C., Hirsh, H., Cohen, W. (1998). Recommendation as Classification: Using Social and Content-Based Information in Recommendation. In Proceedings of the 1998 National Conference on Artificial Intelligence (AAAI-98), pp. 714-720.

Baum, D. & Spann, M. (2014). The Interplay Between Online Consumer Reviews and Recommender Systems: An Experimental Analysis. *International Journal of Electronic Commerce, 19*(1), 129-162. https://doi.org/10.2753/JEC1086-4415190104

Belkin, N.J. & Croft, W.B. (1992). Information filtering and information retrieval: two sides of the same coin? *Communications of The ACM, 35*(12), 29-38.

Bianchini, M., Maggini, M., Sarti, L., Scarselli, F. (2005) Recursive neural networks for processing graphs with labelled edges: theory and applications. *Neural Networks, 18*, 1040-1050. https://doi.org/10.1016/j.neunet.2005.07.003

Bodoff, D. & Ho, S.Y. (2015). Effectiveness of Website Personalization: Does the Presence of Personalized Recommendations Cannibalize Sampling of Other Items? *International Journal of Electronic Commerce, 20*(2), 208-235. https://doi.org/10.1080/10864415.2016.1087821

Breese, J., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pp. 43–52.

Dash, A., Zhang, D., & Zhou, L. (2021). Personalized Ranking of Online Reviews Based on Consumer Preferences in Product Features. *International Journal of Electronic Commerce, 25*(1), 29-50. https://doi.org/10.1080/10864415.2021.1846852

Dong, Z., Wang, Z., Xu, J., Tang, R., & Wen, J. (2022). A Brief History of Recommender Systems. *arXiv preprint*. https://doi.org/10.48550/arXiv.2209.01860

Gao, C., Zheng, Y., Wang, W., Feng, F., He, X., Li, Y. (2022). Causal Inference in Recommender Systems: A Survey and Future Directions. *arXiv preprint*. https://doi.org/10.48550/arXiv.2208.12397

Goldberg, D., Nichols, D., Oki, B.M., & Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of The ACM, 35*(12), 61-70.

Harper, F.M. & Konstan, J.A. (2015). The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems, 5*(4), 1-19. https://doi.org/10.1145/2827872

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.S. (2017). Neural Collaborative Filtering. *Proceedings of the 26th International Conference on World Wide Web, WWW '17* (pp. 173–182). https://doi.org/10.1145/3038912.3052569

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020). LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '20* (pp. 639–648). *https://doi.org/10.1145/3397271.3401063*

Hug, N. (2020). Surprise: A Python library for recommender systems. *Journal of Open Source Software, 5*(52), 2174. https://doi.org/10.21105/joss.02174

Kembellec, G. & Chartron, G. (2014). General Introduction to Recommender systems. In G. Kembellec, G. Chartron, & I. Saleh (Eds.), *Recommender systems* (pp. 1-23). John Wiley & Sons, Incorporated.

Kim, J.W., Lee, B.H., Shaw, M.J., Chang, H.L., & Nelson, M. (2001). Application of Decision-Tree Induction Techniques to Personalized Advertisements on Internet Storefronts. *International Journal of Electronic Commerce, 5*(3), 45-62. https://doi.org/10.1080/10864415.2001.11044215

Kim, K. & Ahn, H. (2011). Collaborative Filtering with a User-Item Matrix Reduction Technique. *International Journal of Electronic Commerce, 16*(1), 107-128. https://doi.org/10.2753/JEC1086-4415160104

Kingma, D.P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. ICLR 2015 Conference, San Diego, CA, USA. https://doi.org/10.48550/arXiv.1412.6980

Kipf, T.N., & Welling, M. (2017). *Semi-Supervised Classification with Graph Convolutional Networks* [Poster presentation]. ICLR 2017 Conference, Toulon, France. https://doi.org/10.48550/arXiv.1609.02907

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer, 42*(8), 30-37. https://doi.org/10.1109/MC.2009.263

Lahtinen J., Martinsen T., & Lampinen J. (2000). Improved rotational invariance for statistical inverse in electrical impedance tomography. *International Joint Conference on Neural Networks, Como, Italy, 2*, 154–158. https://doi.org/10.1109/IJCNN.2000.857890

Lee, J. & Hong, I.B. (2019). Consumer's Electronic Word-of-Mouth Adoption: The Trust Transfer Perspective. *International Journal of Electronic Commerce, 23*(4), 595-627. https://doi.org/10.1080/10864415.2019.1655207

McNee, S.M. Riedl, J., Konstan, J.A. (2006). Being Accurate is Not Enough: How Accuracy Metrics have hurt Recommender Systems. *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, 1097–1101. https://doi.org/10.1145/1125451.1125659

Nguyen, T. & Hsu, P.F. (2022). More Personalized, More Useful? Reinvestigating Recommendation Mechanisms in E-Commerce. *International Journal of Electronic Commerce, 26*(1), 90-122. https://doi.org/10.1080/10864415.2021.2010006

Nikolaeva, R. & Sriram, S. (2006). The Moderating Role of Consumer and Product Characteristics on the Value of Customized On-Line Recommendations. *International Journal of Electronic Commerce, 11*(2), 101-123. https://doi.org/10.2753/JEC1086-4415110205

Nordström, E., Gällmo, O., Asplund, L., Gustafsson, M., & Eriksson, B. (1994). Neural networks for admission control in an ATM network. In L.F. Niklasson, & M.B. Bóden (Eds.), *Connectionism in a broad perspective* (pp. 239–250). Ellis Horwood.

Oestreicher-Singer, G. & Sundararajan, A. (2012). The Visible Hand? Demand Effects of Recommendation Networks in Electronic Markets. *Management Science, 58*(11), 1963-1981. http://dx.doi.org/10.1287/mnsc.1120.1536

Pang, J. & Qiu, L. (2016). Effect of Online Review Chunking on Product Attitude: The Moderating Role of Motivation to Think. *International Journal of Electronic Commerce, 20*(3), 355-383. https://doi.org/10.1080/10864415.2016.1121763

Park, S.H. & Han, S.P. (2013). From Accuracy to Diversity in Product Recommendations: Relationship Between Diversity and Customer Retention. *International Journal of Electronic Commerce, 18*(2), 51-72. https://doi.org/10.2753/JEC1086-4415180202

Piatetsky, G. (2007). *Interview with Simon Funk*. KDnuggets. https://www.kdd.org/exploration_files/simon-funk-explorations.pdf

Qian, N. (1999). On the momentum term in gradient descent learning algorthms. *Neural Networks, 12*, 145-151. https://doi.org/10.1016/S0893-6080(98)00116-6

Ravanifard, R., Mirzaei, A., Buntine, W., & Safayani, M. (2021). Content-Aware Listwise Collaborative Filtering. *Neurocomputing, 461*, 479-493. https://doi-org.ezproxy.uwsp.edu/10.1016/j.neucom.2021.08.076

Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). BPR: Bayesian Personalized Ranking from Implicit Feedback. *Proceedings of the Twenty-Fifth*

Conference on Uncertainty in Artificial Intelligence, UAI 2009 (pp. 452-461). https://doi.org/10.48550/arXiv.1205.2618

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, CSCW '94* (pp. 175-186). https://doi.org/10.1145/192844.192905

Scarselli, F., Tsoi, A.C., Hagenbuchner, M., & Noi, L.D. (2013). Solving graph data issues using a layered architecture approach with applications to web spam detection. *Neural Networks, 48*, 78-90. http://dx.doi.org/10.1016/j.neunet.2013.07.007

Schafer, J.B., Konstan, J.A., & Riedl, J. (2001). E-Commerce Recommendation Applications. *Data Mining and Knowledge Discovery, 5*, 115-153. https://doi.org/10.1023/A:1009804230409

Sperduti, A. & Starita, A. (1997). Supervised Neural Networks for the Classification of Structures. *IEEE Transactions on Neural Networks, 8*(3), 714-735. https://doi.org/10.1109/72.572108

Tang, H., Liu, B., & Qian, J. (2021). Content-based and knowledge graph-based paper recommendation: Exploring user preferences with knowledge graphs for scientific paper recommendation. *Concurrency and Computation Practice and Experience, 33*. https://doi.org/10.1002/cpe.6227

Tsekouras, D. (2017). The Effect of Rating Scale Design on Extreme Response Tendency in Consumer Product Ratings. *International Journal of Electronic Commerce, 21*(2), 270-296. https://doi.org/10.1080/10864415.2016.1234290

University of Minnesota. (2013, April 22). *Fine Tuning the Social Web: John Riedl*. https://research.umn.edu/inquiry/post/fine-tuning-social-web-john-riedl

Villalba-Díez, J., Molina, M., Ordieres-Meré, J., Sun, S., Schmidt, D., & Wellbrock, W. (2020). Geometric Deep Lean Learning: Deep Learning in Industry 4.0 Cyber-Physical Complex Networks. *Sensors, 20*(3). https://doi.org/10.3390/s20030763

Wang, F., Wang, M., Zheng, Y., Jin, J., & Pan, Y. (2021a). Consumer Vigilance and Choice Overload in Online Shopping. *International Journal of Electronic Commerce, 25*(3), 364-390. https://doi.org/10.1080/10864415.2021.1943189

Wang, Q., Du, W., Ma, J., & Liao, X. (2019). Recommendation Mechanism for Patent Trading Empowered by Heterogeneous Information Networks. *International Journal of Electronic Commerce, 23*(2), 147-178. https://doi.org/10.1080/10864415.2018.1564549

Wang, S., Liang, H., Wang, Y., He, X., Sheng, Q.Z., Orgun, M.A., Cao, L., Ricci, F., & Yu, P.S. (2021b). Graph Learning based Recommender Systems: A Review. *arXiv preprint*. https://doi.org/10.48550/arXiv.2105.06339

Wang, X., He, X., Wang, M., Feng, F., & Chua, T.S. (2020). Neural Graph Collaborative Filtering. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '19* (pp. 165-174). https://doi.org/10.1145/3331184.3331267

Weng, S., Liu, S., & Wu, T. (2011). APPLYING BAYESIAN NETWORK AND ASSOCIATION RULE ANALYSIS FOR PRODUCT RECOMMENDATION. *International Journal of Electronic Business Management, 9*(2), 149-159.

Wu, S., Sun, F., Zhang, W., Xie, X., & Cui, B. (2022). Graph Neural Networks in Recommender Systems: A Survey. *ACM Comput. Surv., 37*(4). https://doi.org/10.1145/1122445.1122456

Yuan, X., Han, L., Qian, S., Xu, G., & Yan, H. (2019). Singular value decomposition based recommendation using imputed data. *Knowledge-Based Systems, 163*, 485-494. https://doi.org/10.1016/j.knosys.2018.09.011

Yu, W., Zhang, Z., & Qin, Z. (2022). Low-Pass Graph Convolutional Network for Recommendation. *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence*, pp. 8954-8961.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2021). Graph neural networks: A review of methods and applications. *AI Open, 1*, 57-81. https://doi.org/10.1016/j.aiopen.2021.01.001

Appendix

Data Source

100k and 1M MovieLens data was sourced from the following website, attributable to Harper & Konstan (2015): https://grouplens.org/datasets/movielens/.

Code for Analysis

All code for performing the analysis using MovieLens Data can be found on my public GitHub repository at https://github.com/bfenlon/masters_thesis. Some small changes are required to run the notebooks on a local machine, rather than Google Colab. Additionally, in each Jupyter Notebook file there are changes to be made depending on what dataset is being used. Hyperparameters and data loading are slightly different between the two, and for LightGCN an additional file – lightgcn.yaml – is required to initiate the hyperparameters in the Jupyter Notebook (it is included in the repository).