Abstract

With the wide-spread use of media platforms and entertainment providers, consumers seek to find personalized recommendations, such as for movies, that will match their individual taste. Through the use of recommender systems, and the MovieLens 1M Dataset, we aim to provide a suitable movie recommendation for users based on their own previous movie ratings and other users' ratings. We focus on three models, a Non-negative Matrix Factorization (NMF) model, an Item2Vec model, and a Self-Attentive Sequential Recommendation (SASRec) model, in order to solve this problem. We compare the model performances based on their Normalized Discounted Cumulative Gain (NDCG@10) values, where a higher value signifies that the model gave higher quality recommendations in its ranking of the top 10 most relevant movie items.

- Background: Introduce the problem you are solving, and why is this problem important in practical applications.

All around us, companies and services seek to provide us, the consumers, the best care by recommending products that they believe we would most like. By learning our interests, other items that we disliked and liked, and more information about us, companies can make fairly accurate product recommendations. An example of this would be movie recommendations, with companies like Netflix or Hulu providing personalized movie recommendations based on both movies a user has seen and rated in the past, and movies that other users have seen. This is an example of a collaborative filtering technique, where a model can create relationships between users and items, in this case movies, and analyze different user-item pairs across different users [4, 7]. The problem we aim to solve is to provide users an item recommendation of a movie based on their own previous movie ratings and other users' ratings, with the hopes of the user being interested and liking this movie recommendation. To work on this problem, we will be using the MovieLens 1M Dataset, which consists of 1 million ratings from 6000 users on 4000 movies, where each user has at least 20 movie ratings. In a practical setting, this is important as providing good movie recommendations helps to ensure that a customer or user is satisfied with the service, will develop trust and continue to use the service, and may even recommend the service to friends, ultimately leading to a case where both the user and company benefit.

- Related work: Summarize the papers, articles, and online resources you have read in this class.
"Matrix factorization techniques for recommender systems": This paper discusses two different approaches to recommendation systems, content filtering and collaborative filtering, some examples of each, and what some of the pros and cons of each system are. Content filtering uses profiles made for either users or items, which contain information that can be used to compare items based on similar properties. Collaborative filtering recommends items by looking at past data, such as items rated highly by similar users, so it uses a combination of users and items in order to provide a recommendation through the use of a user-item utility matrix, without

needing explicit information about the users or items themselves. The paper goes on to discuss matrix factorization, where factors for both users and items are taken, such as explicit user ratings for different movies, creating vectors for each. The user-item ratings matrix regularizes the ratings and uses dimensionality reduction to help estimate for blank entries. It also mentions the use of biases and other input sources in order to help deal with problems such as cold start, where the model struggles to make predictions for new users or users with few ratings. Finally, the paper discusses the Netflix Prize competition, and their model's results [4].

"Real-time Personalization using Embeddings for Search Ranking at Airbnb": This paper examines the challenge of optimizing both user and host preferences for Airbnb. The problems that they must consider include cold start, limitation of one guest within a time frame, and understanding guests' short-term and long-term interests. Embedding models and pairwise regression of positive utilities for bookings with negative utilities for rejections were deployed to optimize ranking for both sides of the marketplace. Some specific models used in this research are the skip-gram model and lambda rank [4] model. The train/test process was a multistep process that included preprocessing 800 million click session data, learning 4.5 million embeddings, and tuning various hyperparameters, such as context window size and dimensionality. The results of the study prove that rankings for bookings improved overall with the new methods [3].

"Item2vec: neural item embedding for collaborative filtering": Item2Vec utilizes skip gram with negative sampling (SGNS) to capture the relation between a word to its surrounding words in a sentence. Due to the computational complexity of the softmax function we usually use negative sampling to figure out the number of negative examples to be taken per a positive example. This paper talks about the mathematical derivations for the skip-gram model with negative sampling. The dataset used in this paper is retrieved from Xbox Music Service, to which item2vec is applied. They run the algorithm for 20 epochs, with a negative sampling value of N=15 and dimension parameter m=100. A notable visual that this paper describes is the t-SNE versus SVD embeddings for the item vectors that shows how well the models were able to distinguish the different genres of music. Comparing and contrasting the two models, they study concludes that "item2vec produces a better representation for items than the one obtained by the baseline SVD model" [5].

"Self-Attentive Sequential Recommendation": The other paper we read and implemented is the Self-Attentive Sequential Recommendation paper by Kang and McAuley. Inspired by the Transformer system, the Self-Attentive Sequential Recommendation (SASRec) system was created in an attempt to combine the strengths of Markov Chains (MCs) and Recurrent Neural Networks (RNNs). Both MCs and RNNs have the ability to process sequential data, unlike other kinds of recommender systems. MCs predict the next action based on the last of last few actions, so they typically perform better with sparse datasets. RNNs on the other hand predict the next action based on all or almost all of the previous actions, so they usually perform better with dense datasets. MCs are also better when looking for a simpler model with less parameters, which sacrifices accuracy for ease and less time and money. RNNs are more complex and have longer execution times, but are typically more accurate. SASRec ideally wants to be able to work well regardless of the density of the datasets. This is accomplished by feeding the input data through one or more self-attention blocks, which are composed of a self-attention layer and a point-wise feedforward network. Each iteration through a self-attention block adaptively assigns weights to previous items/actions that capture sequential patterns. This allows SASRec to focus on more recent actions with sparse datasets and a longer range of activity with denser datasets. SASRec was tested on four datasets: Amazon Beauty, Amazon Games, Steam, and MoveLens-1M. Overall, SASRec performed very well compared to other recommender systems in both efficiency and accuracy [6].

<u>- Methodology: The models you have applied to solve the problem. Show technical details and your understanding to the models and model training process.</u>

NMF: Non-negative matrix factorization (NMF) involves first creating a utility matrix of users and ratings and filling in each non-rating as a zero. Then, NMF decomposes this non-negative matrix into two non-negative matrices. The product of those two matrices should closely approximate the original utility matrix, and predicted ratings for previously unrated movies can be extracted. One important parameter for NMF is the number of components. In our movie recommendation system, each component is essentially a genre. Originally, we made the mistake of setting the number of components to the number of movies, which is illogical because there are not anywhere near that many movie genres. As shown later on, increasing the number of components to too many can actually decrease performance. Additionally, it is important to consider the tolerance level, which dictates the stopping condition. By default, alpha_W and alpha_H are often set to zero, which does not allow for regularization of the decomposed matrices; this parameter was also experimented with.

Item2Vec: Item2Vec utilizes skip gram with negative sampling (SGNS) to capture the relation between a word to its surrounding words in a sentence. Due to the computational complexity of the softmax function we usually use negative sampling to figure out the number of negative examples to be taken per a positive example. Some hyperparameters are batch size and embedding size. The batch_size parameter specifies the number of user-item interactions (pairs of user-item interactions) processed in each training iteration. It controls how many samples are used in each forward and backward pass during training. The embedding size refers to the dimensionality of the latent embeddings learned for users and items. In the SGNS method, the model iterates over each sequence in the train_seqs dataset. Each sequence represents a list of items (e.g., words or items in a sequence of interactions). If discard is set to True, it filters out items from the sequence based on a probability threshold determined by prob_discard. Discard is used to expedite the learning process and to improve the representation of rare words significantly. For each item in the sequence, it generates positive samples by considering its neighboring items within a certain context window. It iterates over the context window around the current item index.it creates a positive sample tuple (target, context) and appends it to sgns_samples, labeled as 1. For each positive sample, it generates negative samples by randomly selecting items from the vocabulary that are not in the context list. These samples are used to train the SGNS for learning embeddings, where the model learns to predict the context items given a target item and distinguishes between positive and negative samples during training. We used the package LibRecommender, which is a hybrid recommender system, which means you can choose whether to use features other than user behaviors or not. It supports training for both explicit and implicit datasets, as well as negative sampling on implicit data. It also supports cold-start prediction and recommendation [2].

SASRec: As aforementioned, Self-Attentive Sequential Recommendation's (SASRec) core feature is its self-attention blocks. The SASRec system first converts the user data into a fixed-length sequence that is then input into an item embedding matrix. Each user and item is represented as an embedding vector. This is then injected with a learnable position embedding to preserve the sequential order of the interactions. This complete matrix is fed into a self-attention block, which uses softmax and ReLU functions to calculate the adaptive weights. These weights are supposed to capture the relevance and dependencies between the different items/actions. Based on the model parameters, the matrix can be rerun through additional self-attention blocks. Typically, more self-attention blocks lead to more accurate predictions. However, iterating through too many blocks can result in overfitting, losing gradients, and taking longer to run. In addition to layer normalization, a dropout parameter can be added to the model to remove training neurons when their probability drops below a specified threshold, which reduces the overfitting chance. Following these adjustments, the matrix is fed into an output/prediction layer that determines the relevance of each item in relation to the user. By ranking the items by relevance, the model can predict the next item [1].

- Experiment results.

NMF:

Default Model:

Component	Size
Number of Components	100
Tolerance	0.0001
Max Iterations	200
alpha_W=alpha_H	0

Ablation Table:

Architecture	NDCG@10
Default	0.63434
Number of Components = 4	0.69108
Number of Components = 400	0.63855
Tolerance = 0.05	0.63242
Max Iterations = 10000	0.63455
alpha_W=alpha_H = 0.1	0.81604
alpha_W=alpha_H = 0.001	0.65102
alpha_W=alpha_H=0.001, Max Iterations = 10000	0.65527

For hyper parameter tuning, the NMF models tended to consistently have NDCG@10 scores of around 0.63. When adjusting the number of components, increasing the default 100 to other values, such as 400, showed very little improvement, while decreasing the number of components to very small numbers, such as 1 to 10, showed larger improvements in NDCG@10 scores, but ultimately produced worse movie predictions. When adjusting the tolerance, a slightly larger tolerance of 0.05, or more, slightly lowered the NDCG@10 score, as the model would stop running earlier once it hit the threshold. When adjusting the maximum number of iterations, a larger number slightly improved the NDCG@10, but this is likely from variations when running the code. Finally, when adjusting the alpha_W = alpha_H values, using 0.1 drastically improved the NDCG@10 to about 0.81, but the resulting model predicted the same movie for every user due to this parameter shrinking the utility matrix values to nearly zero. A smaller value of 0.001 was better as they used gently regularized factorization and still improved the NDCG@10 score to about 0.65, but gave reasonable movie predictions. Overall, when combining the results of this hyper parameter tuning, the best NMF model, based on an NDCG@10 of 0.65527, has 100 components, a tolerance of 0.0001, 10000 as the maximum number of iterations, and alpha_W and alpha_H values of 0.001.

Item2Vec:
Default Model:

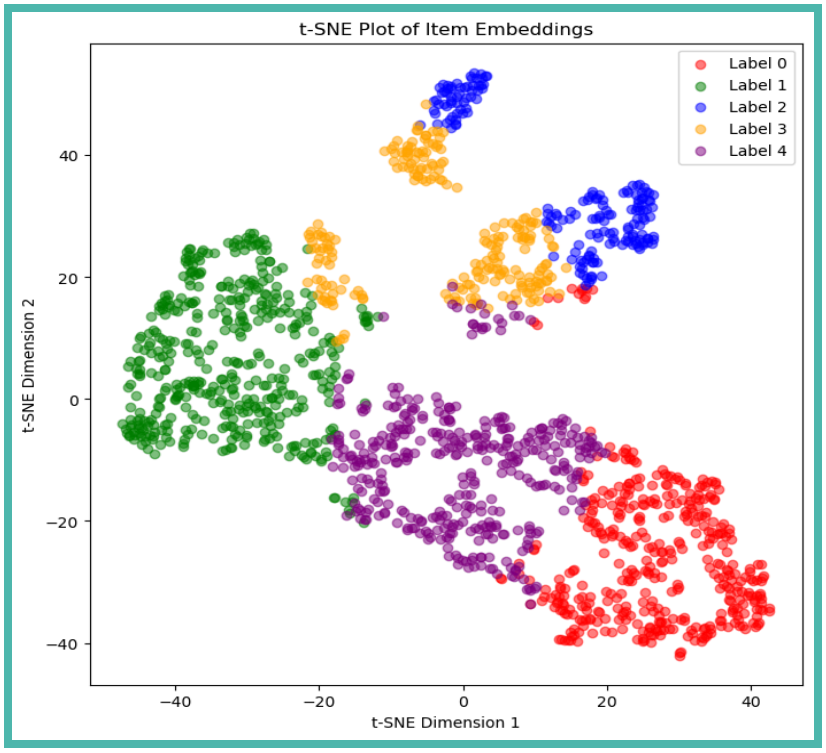
Component	Size
Embedding Size	16
Batch Size	256
Learning Rate	0.001

Epochs	3

Ablation Table:

Architecture	NDCG@10
Default	0.3843
Embedding Size = 30	0.442
Embedding Size = 60	0.450
Batch Size = 64	0.424
Batch Size = 256	0.446
Learning Rate = 0.01	0.446
Learning Rate = 0.001	0.384
Epochs = 10	0.446
Epochs = 20	0.446

The best Item2Vec model, with an NDCG@10 of 0.45, ended up being one with an embedding size of 60, a batch size of 256, a learning rate of 0.01, and 10 epochs. Overall, the hyperparameter turning for the Item2Vec model did not show much promise as the NDCG score remained relatively the same despite some of the changes we tried. For example, increasing the epochs from 10 to 20 did not change the NDCG at all. When working with epochs, at a certain threshold, the NDCG of the model would stop increasing and start to go down (parabolic shape). The most notable hyperparameter tuning for the Item2Vec model was the increase in NDCG from 0.384 to 0.446 when the learning rate was increased. However, 0.01 was the threshold for the learning rate, as increasing it further would make the model perform worse.



t-SNE (t-distributed Stochastic Neighbor Embedding) is a technique for dimensionality reduction that is particularly well-suited for visualizing high-dimensional data in a lower-dimensional space (typically 2D or 3D). Here, we reduce the dimensionality of the item embeddings from their original dimensionality to 2D using t-SNE, based on the Item2Vec model. The graph shows the distributions of labels (1-5) given to the movies.

Graph steps:
1. Got the item embeddings from the lightgcn model. These embeddings represent the learned latent representations of the items in a lower-dimensional space.
2. Used k-means to these item embeddings. We fit the KMeans model to the t-SNE transformed embeddings, and each embedding is assigned a cluster label.
3. Each point in the scatter plot represents an item embedding, and its position is determined by the two t-SNE dimensions. We iterate over each cluster label, extract the indices of embeddings belonging to that cluster, and plot them with the corresponding color

SASRec:
Default Model:

Component	Size
Batch Size	128

Epochs	201
Dropout rate	0.2
Blocks	2

Ablation Table:

Architecture	NDCG@10
Default	0.5886
No Dropout	0.5468
0 Blocks	0.4821
3 Blocks	0.5912
Fewer Epochs (101)	0.5834
Smaller Batch Size (64)	0.5878

The default model here performed very well, with one of the highest NDCG@10. Removing dropout, which helps with overfitting by randomly "turning off" neurons in the model, makes the model perform worse. Having 0 blocks removes most of the self-attention benefits of the model, meaning it only focuses on the last . Having more blocks results in a minor improvement over the default model, and further increasing the number of blocks would likely incrementally improve performance. However, there is a time consideration, since the model using 3 blocks took approximately an hour longer to run than the default model, with only a slight improvement. Decreasing the number of epochs slightly decreased the NDCG@10, as did decreasing the batch size.

Model Comparison:

Model	Best NDCG@10
NMF	0.65527
Item2Vec	0.45
SASRec	0.5912

Conclusion

For NMF, our best performing model had 100 components, a tolerance of 0.0001, 10000 as the maximum number of iterations, and α_W and α_H values of 0.001. For Item2Vec, our best performing model had an embedding size of 60, a batch size of 256, a learning rate of 0.01, and 10 epochs. For SASRec, our best performing model had a batch size of 128, 201 epochs, a dropout rate of 0.2, and 3 blocks.

For a future exploration of this topic, it would be interesting to apply LLMs (Large Language Models) to this challenge. This might be done by giving a list of how a user has rated movies in the past, and then asking the LLM to predict from a predetermined list which movie(s) that the user would like most. It may also be useful to incorporate additional user characteristics, such as their gender, age, and occupation, in order to help provide recommendations for "new" users, or those with minimal data. Besides this, it could be helpful to use the movie genre or category itself as a stronger part of the recommendation process. Additionally, while NMF was our best model, it would be nice to explain our Item2Vec and SASRec models using the Shapley value as well. For further explanation, applying the t-SNE visualization technique to each model to group the movie recommendations by genre instead of by rating could provide some useful insights. Finally, our classmates who also took on the recommender systems challenge used different models than us, so exploring those as well would lead to a more in-depth overview of the topic.

- References

[1]
https://github.com/kang205/SASRec
[2]
https://github.com/massquantity/LibRecommender/blob/master/README.md

[3] Grbovic, Mihajlo, and Haibin Cheng. "Real-time personalization using embeddings for search ranking at airbnb." Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. 2018.

[4] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009): 30-37.

[5] Barkan, Oren, and Noam Koenigstein. "Item2vec: neural item embedding for collaborative filtering." 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP). IEEE, 2016.

[6] Kang, Wang-Cheng, and Julian McAuley. "Self-attentive sequential recommendation." 2018 IEEE international conference on data mining (ICDM). IEEE, 2018.

[7] He, Xiangnan, et al. "Neural collaborative filtering." Proceedings of the 26th international conference on world wide web. 2017.