Movie Recommendation: A Use Case of Bipartite Link Prediction

Presented for the McGill University COMP596 – Network Science Course

Matthew Lesko-Krleza
School of Computer Science
McGill University
Montreal, Quebec, Canada
matthew.lesko-krleza@mail.mcgill.ca

ABSTRACT

Predicting the existence of links in network graphs is a fundamental problem in network science. In the link prediction problem, we are given a graph with a set of nodes and edges. With this graph, we would like to infer interactions between nodes which are likely to occur, or which are missing. Even though this problem has been extensively studied, we would like to demonstrate its use in recommendation systems and compare algorithms that make use of only the graph structure with those that additionally use node attributes.

Content recommendation systems have been increasingly successful in improving user retention and engagement in online services such e-commerce and media streaming. We are concerned with movie recommendation, where our data is used to construct a graph structure of user and movie nodes, and we would like to infer new links and ratings between users and movies. In this paper, we present the application of two existing link prediction algorithms for movie recommendation and rating prediction on the MovieLens-1M (ML-1M) dataset. Our methods demonstrate results that are just under the performance of the state-of-the-art.

KEYWORDS

Recommendation system, movie-user graph network, bipartite graph, link prediction

1 INTRODUCTION

A pervasive challenge in network analysis is the prediction of new edges in a given graph at some point in the future. Having the ability to predict new edges with high accuracy has several beneficial use cases including predicting drug interactions [1], mapping out terrorist cells [2], and predicting new connections in social networks [3]. Link recommendation is a variant of link prediction, in which recommendations such as new 'Facebook' friends, movies, or songs are recommended to a user.

Current link prediction research focuses on link prediction in homogeneous graphs. We'd like to explore and perform link prediction on bipartite graphs with heterogeneous nodes. We apply link prediction algorithms on a bipartite graph created from the MovieLens-1M dataset. This graph is created by connecting users to movies by the reviews the users made for the movies they watched. Predicting links in this graph would effectively be recommending movies to users. We'd like to see how we could take advantage of the network structure and node features to provide movie recommendations.

Existing movie recommendation systems trained on the Netflix and MovieLens datasets approach the problem directly as a machine learning problem – where there is a set of labelled examples, and a model is trained on the examples to discriminate future instances of similar examples – but not as a link prediction problem. In this project, we explore a new way to describe and solve the problem of recommending movies as that of a network science problem. The MovieLens-1M dataset has not been modeled as a graph before, in this project, we do so. We model the MovieLens-1M dataset as a bipartite graph and formulate the movie recommendation challenge to that of performing link prediction for dropped links in the graph. We explore the application of an algorithm that uses only the graph structure and one that uses both the graph structure and node attributes for link prediction, so that we can highlight the effectiveness of learning only from the graph structure and how it can be improved with node features.

We apply an unsupervised link prediction algorithm: random walks, that takes advantage of the network's structure and a supervised algorithm: HinSAGE, a heterogeneous version of GraphSAGE, that takes advantage of both the network's structure and node features to perform link prediction in heterogeneous graphs with nodes.

2 MOTIVATION

Recommendation systems are a lucrative technology because of how they can help internet companies recommend products to users [4]. Amazon.com and Yahoo! recommend documents and products to consumers. Recommendation systems can improve user experience and increase company profit [5] by distilling the immense amounts of information and products to a personalized set of content and items for users. This allows users to read news that's relevant to them or make more desirable purchases. It's needless to say that because of our constant connection to the internet, recommendation systems have become prevalent and are part of our daily lives [6]. Finally, stating content recommendation as a link prediction problem in network science can help expand the work being done in link prediction in network science.

We use the unsupervised random walk because of its ability to learn only from a graph's structure, it's a straightforward algorithm which will be useful to compare with more complex approaches and because we'd like to explore an unsupervised algorithm's ability in recommendation systems. We use GraphSAGE because it can be extended to the bipartite graph scenario, it can learn from the graph's structure and node features, it can additionally perform rating prediction, it can *inductively* learn new node embeddings, and because we'd like to explore a node embedding framework's ability for movie recommendation. It has already been used in Pinterest's recommendation system, making it a natural choice for this problem [7].

3 RELATED WORK

Link prediction. There has been extensive work on link prediction in general. Liben-Nowell and Kleinberg analyze the effectiveness of algorithms such as Common Neighbors, Jaccard's Coefficient, Adamic/Adar, Preferential Attachment, Katz, PageRank, and SimRank for link prediction on social networks [8]. One challenge of link prediction is developing a model that contains information about node and edge attributes and the network structure. Approaches such as PageRank assign scores to nodes based off stationary distributions of random walks. This approach only captures network structure and ignores node attributes. Another way to build a classifier is to extract node attributes and train a supervised classifier, but this ignores the network structure. Backstrom and Leskovec propose a supervised random walk algorithm that combines the extraction of node attributes and the underlying information of the network structure [3]. Node and edge attributes are used in a supervised way to create edge weights for a random walk. Then predictions are made based off the scores from these random walks. There is other work on bipartite link prediction such as modified distance metric learning from Yamanishi [9], the use of a graph kernel: spectral transformation kernel by Kunegis et al [10], projection-based link prediction by Gao et al. [11] and node embedding techniques such as Node2Vec [12]. Gao et al. presented an algorithm that maps a bipartite network onto a unipartite network called a projected graph. Then based off this projected graph, candidate node pairs are selected, and link prediction is performed only for these selected candidates. This has proven to reduce computation time and achieve superior link prediction results for its time [11].

Node embedding. Node representation algorithms have also been gaining popularity in the last decade. The basic idea behind node embedding approaches is to distill high-dimensional node neighborhood information into low-dimensional dense vector embeddings. These embeddings can then be used for downstream machine learning systems and help in tasks such as clustering, link prediction, and node classification [12]. Node2Vec, GATs, and GraphSAGE are just some examples of node embedding frameworks [12-14]. GraphSAGE, not only learns from node neighborhoods but also from node features, which makes it attractive to use in this project since we have features about users and movies. It's already been used for recommending content on Pinterest [7]. This would be the first time it's used for movie recommendation.

Movie recommendation systems. There has been work on predicting ratings on the MovieLens-1M dataset and similar datasets such as the one from Netflix's competition [15]. Most notably, there has been recent work in movie recommendation prediction using neural-network based collaborative filtering. This method has state-of-the-art performance on the MovieLens-1M dataset [16]. As for the Netflix movie recommendation challenge, the 2009 winner used a Restricted Boltzmann Machine to achieve a root mean squared error of 0.8567 [17].

4 PROBLEM DEFINITION

For this project, we define the problem of movie recommendation as a bipartite link prediction problem in a user-movie rating network graph. We first create a bipartite graph from the MovieLens-1M dataset, where one set of nodes consists of users, the another consists of movies. Weighted edges between users and movies represent the rating a user gave for that movie. An edge weight is a user rating from 1 to 5 out of 5, a higher review is better.

Table 1: ML-1M and ML-10M Dataset Features

Network	Ratings	Users	Movies	Tags
ML-1M	1’000’209	6’040	3’883	None
ML-10M	10’000’054	72’000	10’000	100’000

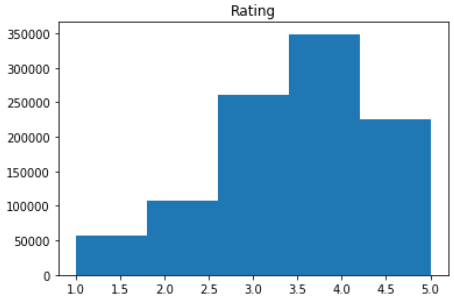

Figure 1: Rating Distribution Histogram

Additionally, we add movie genres and user profile characteristics to their respective nodes, so that we can leverage node attributes to make movie recommendations.

We explore and evaluate the use of an unsupervised algorithm, random walks, and the use of a supervised algorithm, HinSAGE, a heterogeneous variant of GraphSAGE, for link prediction on the user-movie rating graph. The random walks algorithm doesn’t leverage node attributes, whereas HinSAGE leverages both node attributes and the network structure.

3 DATASET DESCRIPTION

Here, we describe the dataset. We’ve added the MovieLens-10M (ML-10M) dataset for comparison. We chose to work with the ML-1M dataset because it’s significantly smaller than its ML-10M equivalent, which will help create and evaluate results quicker. Given the short timeline of this project, quickly coming up with results is vital. Additionally, the ML-1M dataset contains user data, whereas the ML-10M one does not. This data contains information on the user’s age, gender, occupation and demographic location. One thing that is unique for the ML-10M dataset is that users also assigned a ‘tag’ to the movie they reviewed. These can be words such as ‘Excellent!’ or ‘John Travolta.’

ML-1M. The MovieLens-1M dataset is downloaded from the grouplens.org website. All acknowledgements for data collection, and cleaning go to Maxwell Harper et al. [18]. The dataset contains three files for ratings, users and movies.

The ratings file contains weighted user ratings for movies. Each user has given at least 20 movie ratings, and each rating is a number between 1 and 5, where a higher number is a better rating. The movies file contains the titles and genres of each, and every movie reviewed. The genres are chosen from a finite list of 18 genres. Examples of genres are ‘Action,’ ‘Musical,’ and ‘Western.’ Finally, the users file contains users’ gender, age, occupation and demographic location represented as a US zip-code. Demographic location information was optional so not all users have a zip-code available. The occupation was chosen out of a list of 21 possible choices, which included jobs such as ‘artist,’ ‘programmer,’ and ‘college/grad student.’

One can also see in Figure #1, that the ratings are positively skewed. Our intuition is that from this skew, we should expect see a higher tendency to recommend movies than to not. We expect to see a higher recall than precision for link prediction.

4 METHODOLOGY

For this project, we are using the Python 3 programming language and several external Python packages such as Pandas, SciPy, Tqdm, Numpy, Scikit-Learn, Stellargraph, Matplotlib, TensorFlow and Networkx [19-25].

4.1 Graph Creation

For creating the graph, we load movie ratings by chunks of Pandas DataFrame objects to avoid running out of random-access memory (RAM) during graph creation. From the ratings chunk, we extract user and movie identifiers. Then we give them unique node identifiers and a ‘bipartite’ attribute which is set to either ‘u’ for user or ‘m’ for movie. We’ve done this so it’s easier to retrieve the two separate sets of nodes in downstream tasks. Finally, we retrieve the movie ratings from the ratings chunk and assign them as weights to edges between user and movie nodes. We iteratively add the nodes and edges to the graph.

Next, we load and pre-process user and movie features and apply them to their respective nodes. The movies file contains the movies’ titles and genres. Each movie’s list of genres is transformed into a one-hot encoding so that it can be used as a feature for machine learning tasks. Similarly, the users file’s data is loaded and pre-processed. We normalize each user’s age and one-hot encode their gender and occupation. Their zip-code isn’t used because there are numerous users that don’t have one. Finally, the pre-processed movie and user features are applied as features to their respective nodes.

Figure 2 displays the degree distribution plot for the created graph. By analyzing the plot, one can see that there exists anomalies in the graph. Since each user has given at least 20 ratings, the anomalies could be movies that haven’t gotten many ratings, since their degrees are less than 20.

Table 2: Graph Statistics

Property	Value
Number of Nodes	9’746
Number of Edges	1’000’209
Average Degree	205.25
Is Connected?	True

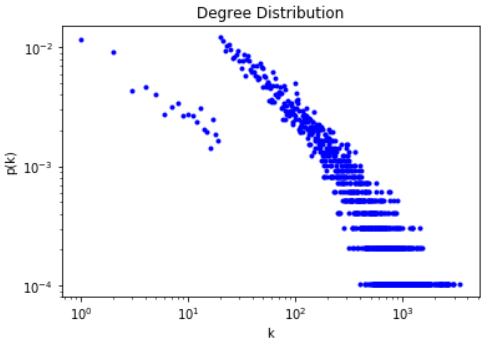

Figure 2: Graph Degree Distribution

Additionally, Table 2 shows that the graph is connected. Each movie can be traversed to by starting at any user. Finally, it looks like there are some movies and/or users that aren’t included in the set of ratings, since the sum of users and movies doesn’t add up to the number of nodes in the graph, when it should be. The authors of the dataset have claimed that there could be errors within it [18].

4.2 Link and Rating Prediction

There is a difference between predicting new links and predicting new movie ratings. In the former case, we only consider predicting the existence or non-existence of links between users and movies. In the latter, we are also concerned with predicting the rating a user gives for a movie.

4.3 Unsupervised Random Walk Algorithm

Next, we describe the unsupervised random walk algorithm. A random walk is a finite stochastic process consisting of a succession of random steps within a graph *G=(V, E)*. This can be modelled as a finite Markov chain. Let each edge *(u, v)* $\in$ *E* assigned a weight *w(u, v)*. Then each edge can be assigned a transition probability $P_{u,v}$= *w(u, v) / d(u)* where *d(u)*= $\Sigma w(u,v)$ is the weighted degree of node *u*. This transition probability is used during a random traversal of the graph. Given the distribution of probabilities $p_t = [p_t(0), \dots, p_t(n)]$ of ending a random walk at a particular node *n*, it’s proven that this distribution converges to a *stationary distribution* p_s as $t \to \infty$ where $p_s = p_s * P$ [26].

So, after several iterations, the distribution of node transitions can be used as scores to determine likeliness of new links. Each score is some probabilistic value between 0 and 1, where 0 is the lack of an edge, whereas 1 represents with 100% certainty that there will be a new edge.

These scores are then used to measure the Receiver Operating Characteristic Area Under-the-Curve (ROC AUC). We detail the experimental setup in Section 5, but to summarize, we drop several edges and perform a random walk starting from the nodes of the dropped edges. The scores from the random walk are ‘predictions’ and the dropped edges are our ground-truth.

We use this algorithm to perform link prediction but not movie rating prediction. The Experiment section details how we evaluated the effectiveness of this algorithm. The Results section details their resulting effectiveness.

4.4 GraphSAGE

Here, we highlight GraphSAGE’s components. GraphSAGE is a framework for *inductive* node embedding. It learns *aggregator functions* that induce a new node’s embedding given its features and local neighborhood [13]. Therefore, it can predict embeddings for new nodes without needing to perform a re-training procedure. As opposed to DeepWalk, a *transductive* algorithm that needs to be re-trained each time a new node is added to the training graph [27]. It does so because it requires an entire graph to learn node embeddings as opposed to a set of aggregator functions. The intuition behind the embedding generation algorithm is for each iteration, nodes aggregate information from their local neighbors, and as this process iterates, nodes incrementally gain more and more information from further reaches of the graph. At each step k, each node $v \epsilon V$ aggregates the representations of nodes and its immediate neighbors into a single vector $h_{N(v)}^{k-1}$. GraphSAGE then concatenates the node’s current representation h_v^{k-1} with the aggregated neighborhood vector $h_{N(v)}^{k-1}$, and this concatenated vector is fed through a fully connected layer with nonlinear activation σ. This transforms the representations to be used for the next step of the algorithm. We will go more into depth about the aggregator functions and the other main components.

There are three components to GraphSAGE: *context construction, information aggregation,* and *loss function.*

Context construction. GraphSAGE has a context-based similarity assumption, meaning that GraphSAGE assumes that nodes that reside in the same neighborhood should have similar embeddings. Each node’s neighborhood’s

depth is parametric as a number of hops N and can be either increased or decreased to either increase or decrease information sharing between nodes.

Aggregation functions. Aggregators take a neighborhood as input to and combine each neighbor's embedding to create a neighborhood embedding. This aggregates information from the node's neighborhood. A forward propagation algorithm is used to generate neighborhood embeddings. This algorithm first initializes embeddings for each node. Then captures a node's neighborhood of depth N, creates a neighborhood embedding and concatenates it with the source node's existing neighborhood embeddings. This concatenated vector is then passed through a neural network layer to update the source node's embedding. The advantage of learning aggregator functions is that unseen node embeddings can be generated from its features and neighborhood. There are three available aggregator functions: a mean aggregator, a Long Short-Term Memory (LSTM) aggregator, and a pooling aggregator.

The mean aggregator takes the elementwise mean of the neighborhood representation vectors $\mathbf{h}_v^k$, where k is the representation at step k and v is a specific node and its neighborhood. W are sets of weight matrices learnt at each step and are used to propagate information between layers of the model. $N(v)$ denotes the immediate neighbors of a node v. The mean aggregator computes the mean of the node representations of node v and its immediate neighbors u:

$$\mathbf{h}_v^k \leftarrow \sigma\left(W * MEAN\left(\mathbf{h}_v^{k-1} \text{ U } \mathbf{h}_u^{k-1}, \forall\, u \in N(v)\right)\right) \quad (1)$$

This is a rough, linear approximation of a localized spectral convolution with weights W.

The LSTM aggregator is based on the LSTM architecture. Compared to the mean aggregator, LSTMs have larger expressive capability. The LSTM is adapted to operate on a random permutation of the node's neighbors. Unfortunately, the authors of the paper don't go into detail about the implementation of this aggregator.

The pooling aggregator feeds each neighbor's vector through a fully-connected neural network and performs a max-pooling operation to aggregate information across the neighbor set:

$$AGGREGATE_k^{pool} = \max(\sigma\left(W_{pool}\, \mathbf{h}_v^k + \mathbf{b}\right), \forall\, u_i \in N(v)) \quad (2)$$

Max is an element-wise max operator and σ is a nonlinear activation function. By applying max-pooling to each of the computed features of the representation, the model effectively captures different aspects of the neighborhood set. Intuitively, the multi-layer perceptron can be thought of as a set of functions that compute features for each of the node representations in the neighbor set.

For this project, we used the pooling aggregator because of its efficiency as stated in the authors' paper.

Loss function. Learning aggregators and embeddings requires a differentiable loss function. Hamilton et al. present the following for computing loss between two nodes *u* and *v*:

$$J_G(\boldsymbol{z}_u) = -\log\left(\sigma(\boldsymbol{z}_u^T \boldsymbol{z}_v)\right) - Q * \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-\boldsymbol{z}_u^T \boldsymbol{z}_{v_n})) \quad (3)$$

The first term rewards maximizing similarity of neighboring nodes *u* and *v*. The second term sets apart embeddings for independent nodes.

We use a variation of GraphSAGE, called HinSAGE, for link and movie rating prediction.

4.5 HinSAGE Model

HinSAGE is the heterogeneous variant of GraphSAGE [19]. In this variation, a constraint is imposed on the original framework. Node neighbors can only be nodes from the opposite side of the bipartite graph and links can only be predicted between nodes from different sets. So, during training, a node's neighbors only consist of nodes different from the source node and during inference, new links are only predicted between user and movie nodes.

This node embedding framework will perform supervised learning using the one-hot encodings of movie genres, one-hot encodings of user occupations and gender, normalized user age, and edge weights/ratings as features. The supervised learning algorithm uses a node's features and neighborhood to create a node embedding. This is the key component; the embedding is used to extract features from new nodes and used either for link prediction or rating prediction.

Link prediction. For link prediction, the final layer of the HinSAGE model is a ReLU activation.

Rating prediction. For rating prediction, the final layer of the HinSAGE model is a linear regression layer.

The Experiment section details how we evaluated the effectiveness of this algorithm. The Results section details its resulting effectiveness.

5 EXPERIMENTS

We test the performance of the Unsupervised Random Walk on the task of (i) predicting dropped links in the MovieLens-1M graph. We also test the performance of GraphSAGE on the task of (i) predicting dropped links and

of (ii) predicting user-movie ratings on the MovieLens-1M graph.

5.1 Random Baselines

For comparison purposes, we have tested a random predictor baseline. Given a set of candidate edges, the predictor randomly assigns a probability to the candidate edge of becoming a new link. This is not meant to be used as an effective predictor. This was implemented from scratch in Python 3. Additionally, we evaluate a non-trained HinSAGE model at rating prediction, since the weights are randomly initialized, this can be intuitively interpreted as a random baseline.

5.2 Unsupervised Random Walk

For evaluating the unsupervised random walk at link prediction, we first drop 20% of edges within the graph. Then we sample the same number of previously non-existent edges between user and movie nodes from the graph. This is done so that we have an equal number of positive and negative links to predict.

We combine these two sets of edges and assign all of their probabilities of an edge to 0, this set of edges are to be used as the candidate edges during the random walk.

We combine the same two sets of edges and assign each edge with a value of either 0, or 1, signifying the non-existence and existence of an edge respectively. This will be used for validating the effectiveness of the random walk. Finally, 10 tests of the unsupervised random walk were performed for measure variance and their performances were averaged out.

This was implemented from scratch in Python 3.

5.3 HinSAGE

For training the HinSAGE model, the following configuration is used:

- Training epochs: 3;
- Batch size: 64;
- Dropout rate [28]: 0;
- Optimizer [29]: Adam;
- Learning rate: $1e^{-2}$;

For evaluating link and rating prediction, we draw all the existing nodes from the graph and split the edges into an 80/10/10 split, where 80% of edges are used for training, 10% for validation, and 10% for testing.

An implemented version of the model was available through the Stellagraph Python package [19]. We trained the model on our data using the model provided from the package. Finally, we train and evaluate HinSAGE performance for three different runs each with a different seed for splitting the train, validation and test sets to have measure variance and average out its metrics. We leave hyperparameter tuning for future work.

5.4 Metrics

Link prediction. To evaluate link prediction effectiveness, we measure ROC AUC.

Rating prediction. To evaluate rating prediction effectiveness, we measure Root Mean Squared Error (RMSE).

6 RESULTS

6.1 Unsupervised Random Walk Link Prediction

The random walk computes for an average of 12 minutes and 53 seconds on a dual-core Intel i5 CPU for 6040 user nodes. When averaged over 10 different runs, each with a different seed for sampling dropped links, the algorithm performs with a score of 0.8453 ROC AUC. Compared to a random baseline of 0.4990 ROC AUC, one can see that this is an effective algorithm which only uses the inherent network structure of the graph. Contrary to our initial hypothesis that the algorithm's result would have a higher recall than precision, the resulting averaged ROC curve in Figure 3 suggests that there isn't any preference for either recall nor precision.

Table 3: Link Prediction Results

Method	ROC AUC
Random Baseline	0.4990
Unsupervised Random Walk	**0.8453**
HinSAGE	0.8033

Table 4: Movie Rating Prediction Results

Method	RMSE
Random Baseline	3.8256
HinSAGE	1.0605
Neural Network CF	**0.8480**

6.2 HinSAGE Link Prediction

When averaged out over 3 separate runs, HinSAGE takes 57 minutes and 27 seconds to train for 3 epochs and achieves a ROC AUC of 0.8033 as denoted in Table 3. Although HinSAGE effectively learns from the graph, as evidenced by the fact that its score beats that of the random baseline, it fails to beat the unsupervised random walk. This is surprising because the model leverages node attributes. Some of the possible reasons why it didn't

achieve a higher performance is the lack of fine-tuning and possibly requiring more node features.

6.3 HinSAGE Rating Prediction

One can see in Table 4 that HinSAGE performs significantly better than the random baseline, but it fails to beat the state-of-the-art neural network based collaborative filtering approach. However, HinSAGE was only trained for 3 epochs with little fine-tuning. We believe that one possible avenue for future work is to achieve the same performance or beat this state-of-the-art method by fine-tuning and training HinSAGE for longer.

6.3 HinSAGE Learnt Representations

Additionally, Figure #5 contains a T-Distributed Stochastic Neighbor Embedding (TSNE) plot of the HinSAGE node embeddings. From analyzing the visualization of user node embeddings, where the colors determine a user's profession, one of the notable node attributes, we can see that there doesn't seem to a strong correlation in between occupation and movie ratings. If there were a strong correlation of seeing users of the 'Programmer' occupation to watch and highly rate sci-fi movies, then there would be a noticeable cluster of nodes of the 'Programmer' occupation color.

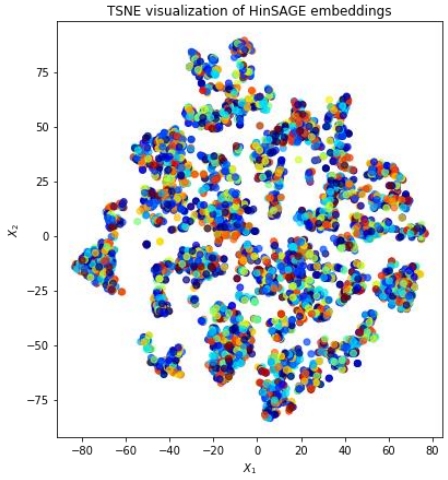

Figure 5: HinSAGE Learnt User Node Representations TSNE Plot

7 CONCLUSION

We applied an unsupervised random walk and a supervised node embedding framework based off GraphSAGE called HinSAGE to perform link and rating prediction. The unsupervised random walk's link prediction performed surprisingly well considering that it only relies on the network structure of the graph.

Additionally, we demonstrate a use case for Graph/HinSAGE since it performs comparably well to the state-of-the-art baseline in movie rating prediction. Possible extensions and improvements include fine-tuning HinSAGE, using an ensemble method of the node embedding model, and developing and evaluating a supervised random walk algorithm. Additional tasks for future work involves testing the two methods on datasets of different sizes such as the MovieLens-100K, MovieLens-10M, MovieLens-20M, the synthetically expanded MovieLens-1B and the 2009 Netflix Prize datasets. Our results prove Unsupervised Random Walk's and HinSAGE's abilities to learn from bipartite user-product graphs and to perform recommendation.

Finally, we believe there is future work where these algorithms can be used for other recommendation systems in different domains, such as e-commerce, internet search, and event recommendation. The main difference is that movie nodes would be replaced by the domain's items to be recommended, and user nodes would user information gathered from the target domain.

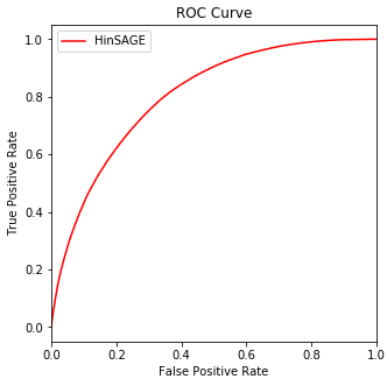

Figure 4: HinSAGE ROC Curve

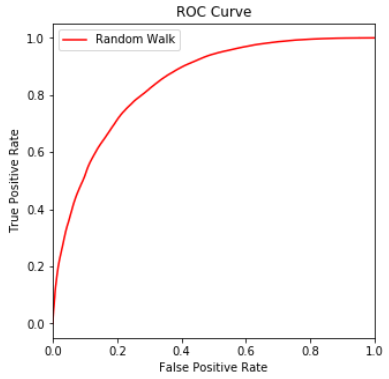

Figure 3: Unsupervised Random Walk ROC Curve

REFERENCES

[1] A. Fokoue, M. Sadoghi, O. Hassanzadeh, and P. Zhang, "Predicting Drug-Drug Interactions Through Large-Scale Similarity-Based Link Prediction," Cham, 2016, pp. 774-789: Springer International Publishing.

[2] V. E. J. C. Krebs, "Mapping networks of terrorist cells," vol. 24, no. 3, pp. 43-52, 2002.

[3] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *Proceedings of the fourth ACM international conference on Web search and data mining*, 2011, pp. 635-644: ACM.

[4] A. Ansari, S. Essegaier, and R. Kohli, "Internet recommendation systems," ed: SAGE Publications Sage CA: Los Angeles, CA, 2000.

[5] A. Das, C. Mathieu, and D. J. a. p. a. Ricketts, "Maximizing profit using recommender systems," 2009.

[6] Z. Wang, X. Yu, N. Feng, Z. J. J. o. V. L. Wang, and Computing, "An improved collaborative movie recommendation system using computational intelligence," vol. 25, no. 6, pp. 667-675, 2014.

[7] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974-983: ACM.

[8] D. Liben-Nowell, J. J. J. o. t. A. s. f. i. s. Kleinberg, and technology, "The link-prediction problem for social networks," vol. 58, no. 7, pp. 1019-1031, 2007.

[9] Y. Yamanishi, "Supervised bipartite graph inference," in *Advances in Neural Information Processing Systems*, 2009, pp. 1841-1848.

[10] J. Kunegis, E. W. De Luca, and S. Albayrak, "The link prediction problem in bipartite networks," in *International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems*, 2010, pp. 380-389: Springer.

[11] M. Gao, L. Chen, B. Li, Y. Li, W. Liu, and Y.-c. J. I. S. Xu, "Projection-based link prediction in a bipartite network," vol. 376, pp. 158-171, 2017.

[12] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855-864: ACM.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024-1034.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. J. a. p. a. Bengio, "Graph attention networks," 2017.

[15] J. Bennett and S. Lanning, "The netflix prize," in *Proceedings of KDD cup and workshop*, 2007, vol. 2007, p. 35: New York, NY, USA.

[16] H. Lee and J. J. I. E. Lee, "Scalable deep learning-based recommendation systems," vol. 5, no. 2, pp. 84-88, 2019.

[17] Y. J. N. p. d. Koren, "The bellkor solution to the netflix grand prize," vol. 81, no. 2009, pp. 1-10, 2009.

[18] F. M. Harper and J. A. J. A. t. o. i. i. s. Konstan, "The movielens datasets: History and context," vol. 5, no. 4, p. 19, 2016.

[19] C. s. Data61, "StellarGraph Machine Learning Library," ed: \url{https://github.com/stellargraph/stellargraph}, 2018.

[20] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265-283.

[21] J. D. J. C. i. s. Hunter and engineering, "Matplotlib: A 2D graphics environment," vol. 9, no. 3, p. 90, 2007.

[22] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," vol. 12, no. Oct, pp. 2825-2830, 2011.

[23] E. Jones, T. Oliphant, and P. Peterson, "SciPy: Open source scientific tools for Python, 2001," ed, 2016.

[24] C. O. J. T. J. o. O. S. S. da Costa-Luis, "tqdm: A Fast, Extensible Progress Meter for Python and CLI," vol. 4, p. 1277, 2019.

[25] A. Hagberg *et al.*, "Networkx. High productivity software for complex networks," 2013.

[26] L. J. C. Lovász, Paul erdos is eighty, "Random walks on graphs: A survey," vol. 2, no. 1, pp. 1-46, 1993.

[27] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701-710: ACM.

[28] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. J. a. p. a. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012.

[29] D. P. Kingma and J. J. a. p. a. Ba, "Adam: A method for stochastic optimization," 2014.