Abstract
Recommender systems are software tools and techniques used to provide content suggestions for items to be of use to a user based on either their prior history or the behavior of users deemed similar to them. The objective of our research was to better understand and develop a proof of concept for probabilistic recommender systems, which use Bayesian approaches to machine learning to improve some of the existing flaws with current recommender systems. Our experiments involved implementing one of such probabilistic recommender systems and performing it on the MovieLens 1M dataset provided by Netflix. While our proof of concept was limited, our work gives hope for future iterations that could be more scalable and more flexible on different datasets.

Introduction
Recommender systems are defined as a “subclass of information filtering system[s] that seek to predict the ‘rating’ or ‘preference’ a user would give to an item” [8]. Use cases include recommending movies for a Netflix user to watch or generating playlists for Spotify listeners.

Modern recommender systems tend to be memory intensive as opposed to compute intensive and exhibit higher storage requirements (GBs) and irregular memory accesses. In addition to this, most are not representative of use cases and do not scale well [3]. Furthermore, existing optimization techniques often cannot be applied to recommendation models as the models are intrinsically different, introducing unique memory and computational challenges. Due to this and its wide variety of commercial applications, there’s interest in optimizing the performance capabilities of modern recommender systems.

This paper is organized as follows. We will first give an overview of more traditional machine learning models that have been used to solve recommendation problems, a brief description of probabilistic machine learning, and then examine current probabilistic recommender systems. The paper will then go into detail about a proof of concept we developed of one such probabilistic recommender system, the latent Dirichlet allocation (or LDA), model. The conclusion of the paper includes descriptions of the testing, results, and lastly a reflection on our experiments.

Recommender Systems
Among these in the current landscape include matrix factorization, multi-layer perceptron, and neural matrix factorization, all of which will be discussed in this paper. The results and performance of these algorithms will be discussed in terms of their usage in experiments performed in a 2017 paper titled “Neural Collaborative Filtering” presented by a group of researchers, He et al., at the 26th International Conference on World Wide Web, and the “Collaborative Topic Modeling for Recommending Scientific Articles” paper written by Dr. Chong Wang and Dr. David Blei of Princeton University’s Computer Science Department.

Matrix factorization (MF) is the de facto approach to latent factor model-based recommendation problems [4]. It projects users and items into a shared latent space, using a vector of latent features to represent a user or an item. After that, a user’s interaction on an item is modeled as

the inner product of their latent vectors. The MF algorithm then associates each user and item with a latent vector and forms a linear model of interactions between both latent factors. The similarity between items is measured by inner product and Jaccard coefficient as the groundtruth similarity of two users that MF needs to recover. Generalized matrix factorization (GMF) is the most standard variant of MF but with an activation function that can either be linear or nonlinear. MF performs best on trained models with random initializations until convergence using Adaptive Moment Estimation, for example, the MovieLens and Pinterest implicit data tested in the paper. However, it can also be used for any recommendation task with a fitting embedding. Per the results in the He et al. paper, MF can at times result in large ranking loss, but one way to resolve the issue is to use a large number of latent factors K, but may adversely hurt the generalization of the model (e.g., overfitting the data), especially in sparse settings. While matrix factorization cannot perform out-of-matrix prediction use cases, matrix factorization works well for in-matrix prediction, but adding content with CTR improves performance. This improvement is greater when the number of returned documents is larger.

The next algorithm of interest is multi-layer perceptron (MLP). This algorithm adopts two pathways to model users and items, and after combining the features of two pathways by concatenating them yields a multi-layer perceptron which uses a tower structure to design the network pattern. As the tower architecture expands, the number of hidden variables decreases, allowing for the learning of abstractive features. Use cases that would take advantage of MLP's strengths include trained models with random initializations until convergence including again the MovieLens and Pinterest implicit datasets. MLP slightly outperforms generalized MF algorithms on the same data as the more layers present in the perceptron, the better performance in general. However, the tradeoff with this improved performance comes with low interpretability and high training time and the possibility of overfitting [4].

The novel recommender system proposed in the He et al. paper is dubbed neural matrix factorization (NeuMF) and serves as a combination of matrix factorization and multi-layer perceptron. It borrows from the idea of matrix factorization by providing embedding of the user and item input to both models separately but also uses the ideas behind multi-layer perceptron by concatenating the output into a final hidden layer [4]. The parameters for this novel system are initialized by independently training MF and MLP from scratch. NeuMF shares the same use cases and was tested using the MovieLens and Pinterest implicit datasets. In the tests performed, NeuMF outperforms state of the art recommendation architectures with significant improvements if the input data is pretrained. There is also hope of future improvements by adding additional layers. As a combination of the strengths of the two previously mentioned recommender systems, it shares similar use cases and was tested on the MovieLens and Pinterest implicit data.

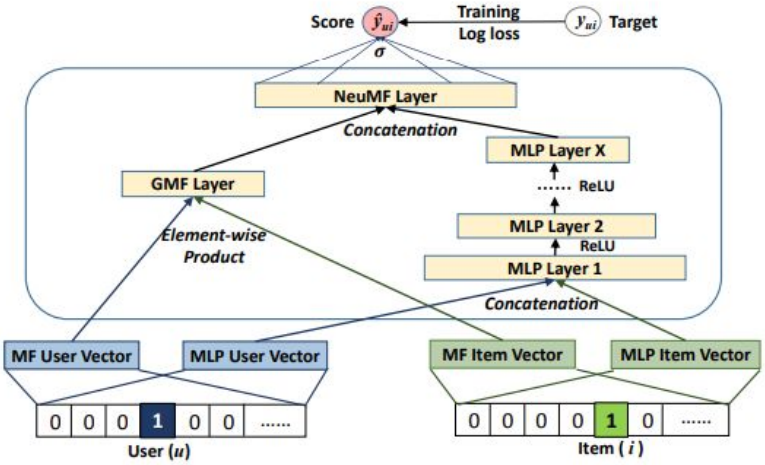

Figure 1: Visual diagram of NeuMF and its overlap with MF and MLP

Probabilistic Machine Learning

While these are traditional recommenders methods, of interest in our research was probabilistic approaches for generating recommendations. The dominant frequentist version of deep learning has many flaws, including being data hungry and compute-intensive, poor at representing uncertainty, easily fooled, lacking transparency, and assuming parameters to be fixed numbers, but probabilistic machine learning avoids these pitfalls [1]. As we looked into the landscape of probabilistic machine learning, there are two large families of structures in the discipline: Bayesian models and Bayesian inference algorithms.

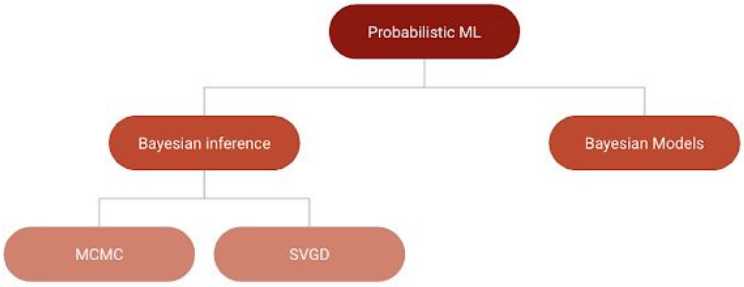

Figure 2: Family of probabilistic machine learning landscape

Bayesian models can be defined as a statistical model where probability can be used to represent all uncertainty within the model, both the uncertainty regarding the output but also the uncertainty inherent to the parameters of the model, solving many of the aforementioned issues with frequentist approaches to machine learning [15]. This is done by allowing the model's parameters to have their own distributions and incorporates additional information outside of what is in the data itself, allowing for context.

There are many Bayesian inference algorithms including Markov Chain Monte Carlo (MCMC), and Stein variational inference, more specifically Stein Variational Gradient Descent (SVGD).

MCMC algorithms, including the famous Metropolis-Hasting algorithm, generate samples from a given probability distribution by setting up a Markov chain with the stationary distribution desired. After this, MCMC algorithms simulate a random walk along the Markov chain's states to continuously generate states to be later used as samples. This sampling process is pretty heavy but has no bias and, so, these methods are preferred when accuracy matters more than speed [10]. Next, SVGD forms a gradient descent for optimization through iteratively transporting input particles to match with the target distribution, by applying a form of functional gradient descent that minimizes the Kullback–Leibler divergence. SVGD is commonly used to solve problems including Variational Inference, Reinforcement Learning, and generative adversarial network or GAN, zero-sum games [5]. For Bayesian inference algorithms in general, the time required to generate the samples is quite large, providing motivation for the next family of probabilistic algorithms.

Probabilistic Recommender Systems

There are examples of Bayesian models that use probabilistic machine learning to solve recommendation problems. Of these, Restricted Boltzmann Machine (RBM), Markov Random Fields (MRF), and Latent Dirichlet Allocation (LDA) are among the most popular.

Restricted Boltzmann Machine (RBM) is a neural network model typically used to perform unsupervised learning. In general, Boltzmann machines are generative deep learning models with only hidden and visible nodes, allowing them to be non-deterministic (or stochastic) in nature [11]. RBMs are a special class of Boltzmann machines with restrictions on the connections between the hidden and visible nodes, making them easier to implement. They form a two-layered (one hidden and one visible) artificial neural network with generative capabilities. The two layers are connected by a bipartite graph making them more efficient to train than general Boltzmann machines. While RBMs are not used as commonly now, they can be used to initiate the weights of a neural network in model training for classification problems and problems that include imbalanced data sets.

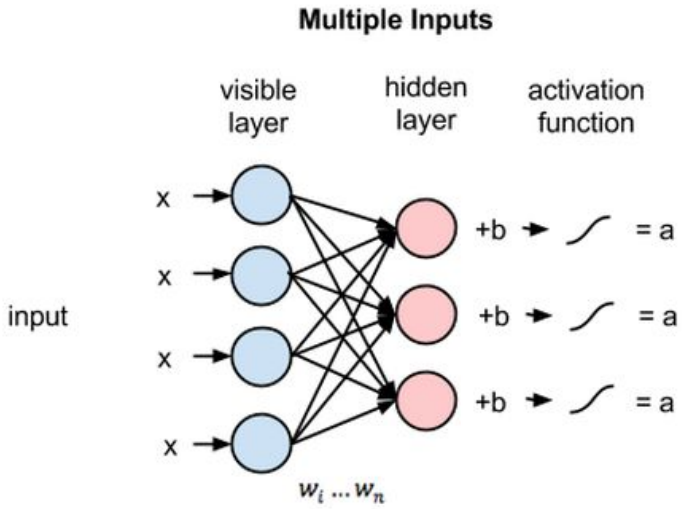

Figure 3: Example bipartite graph that makes up an RBM

The Markov random fields (MRF) model is an undirected network of dependencies that may or may not be cyclic. This allows MRFs to represent certain dependencies that a Bayesian network cannot. MRFs use sparsity to speed up the training by reducing the number of parameters, in essence trading accuracy for training-time [12]. Instead of training each node in the input, MRF takes advantage of sparsity by applying a threshold for covariance and making an overgeneralization to train dense groups at once rather than single nodes. As such, this model is typically used on collaborative-filtering problems with a large number of items such that the inputs are sufficiently sparse. When tested, while requiring several orders of magnitude less training time, the proposed sparse approximation resulted in a model with fewer parameters than the competing models, while obtaining about 20% better ranking accuracy on the data-set with the largest number of items in the experiments. In addition to this, the training time can be reduced from about 16 minutes for the closed-form solution to under a minute with only a relatively small loss in accuracy; different trade-offs between accuracy and training-time can be obtained by using a sparser model and/or a larger hyper-parameter “r” (which increases the sizes of the subsets of items). Typical applications of MRFs include image modelling and computer vision problems.

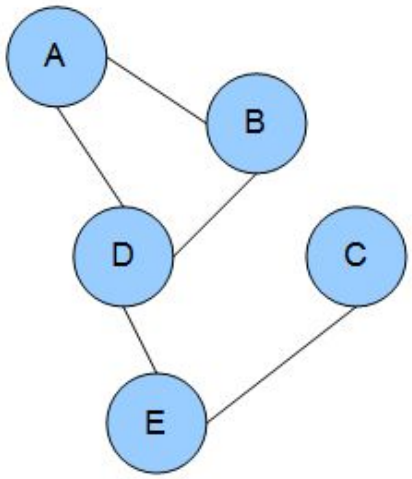

Figure 4: Simple MRF network

Lastly, Latent Dirichlet Allocation (LDA) has the ability to discover latent semantic structures from complex corpora. At a high level, LDA maps all the documents in the input corpus of texts to a list of topics it generates in a way such that the words in each document are mostly captured by those imaginary topics. From there, the model learns the topics, decomposes the documents, and lastly situates content in each document in terms of the topics, in essence assigning each document a similarity value for each topic. LDA lastly will produce an identical document using this process of topic association for each input document. Given the more text and word-centric approach to filtering, the use cases for LDA generally include text document mining, topic modeling, corpi of documents, image classification, video streams, and cleaned textual-based datasets such as the CiteULike dataset. In the testing performed in the Wang and Blei paper, LDA articles with high frequencies tended to have high recalls for in-matrix prediction

and their predictions have less variance but performed worse than other existing methods, such as collaborative filtering, and much worse for out-of-matrix prediction [13].

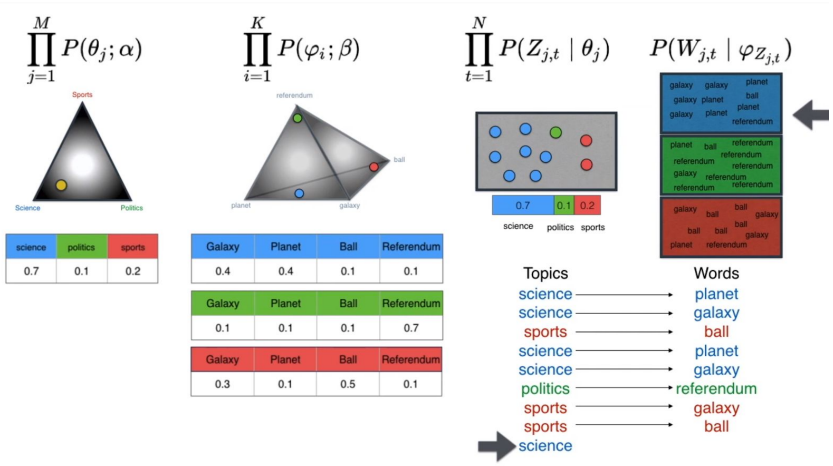

Figure 5: Flow following end-to-end LDA process

Implementation

Our implementation was heavily inspired by the 2014 "Improving Collaborative Filtering based Recommenders using Topic Modelling" paper [14] which also sought to use LDA on non-textual data.

Our implementation of the LDA algorithm on the Netflix MovieLens 1M dataset can be seen at the following repository: https://github.com/festusojo123/MovieLens-LDA.

The MovieLens 1M dataset consists of a list of 4000 movies:

movies

movie_id	movie	genres
1	Toy Story (1995)	Animation|Children's|Comedy
2	Jumanji (1995)	Adventure|Children's|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children's
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action|Adventure|Thriller
11	American President, The (1995)	Comedy|Drama|Romance
12	Dracula: Dead and Loving It (1995)	Comedy|Horror
13	Balto (1995)	Animation|Children's
14	Nixon (1995)	Drama
15	Cutthroat Island (1995)	Action|Adventure|Romance
16	Casino (1995)	Drama|Thriller
17	Sense and Sensibility (1995)	Drama|Romance
18	Four Rooms (1995)	Thriller
19	Ace Ventura: When Nature Calls (1995)	Comedy
20	Money Train (1995)	Action
21	Get Shorty (1995)	Action|Comedy|Drama
22	Copycat (1995)	Crime|Drama|Thriller
23	Assassins (1995)	Thriller

Figure 6: movies.csv file from MovieLens 1M dataset

and the ratings of 6000 users on each of these movies:

ratings

user_id	movie_id	rating	timestamp
1	1193	5	978300760
1	661	3	978302109
1	914	3	978301968
1	3408	4	978300275
1	2355	5	978824291
1	1197	3	978302268
1	1287	5	978302039
1	2804	5	978300719
1	594	4	978302268
1	919	4	978301368
1	595	5	978824268
1	938	4	978301752
1	2398	4	978302281
1	2918	4	978302124
1	1035	5	978301753
1	2791	4	978302188
1	2687	3	978824268
1	2018	4	978301777
1	3105	5	978301713
1	2797	4	978302039
1	2321	3	978302205
1	720	3	978300760
1	1270	5	978300055

Figure 7: ratings.csv file from MovieLens 1M dataset

However, the initial input for an LDA model must be a corpus of texts, so our first step was converting the movie titles in the dataset into a corpus of texts by associating movie titles, genres, plot summaries from a Wikipedia dataset from Kaggle [9] with their corresponding movieIDs, dubbing this ‘synthesizedMovieInfo.csv’. We also used another dataset from Kaggle [7] to fix the issue of invalid name types from MovieLens 1M compared to how the titles are stored in Wikipedia. Next to finish the pre-processing step, we used 'corpus.py' to create ‘updatedCorpus.zip’ which itself contains 3123 movie documents of interest containing genre and summary.

An example of these updated input files can be seen below for the movie with movie_id 1, *Toy Story:*

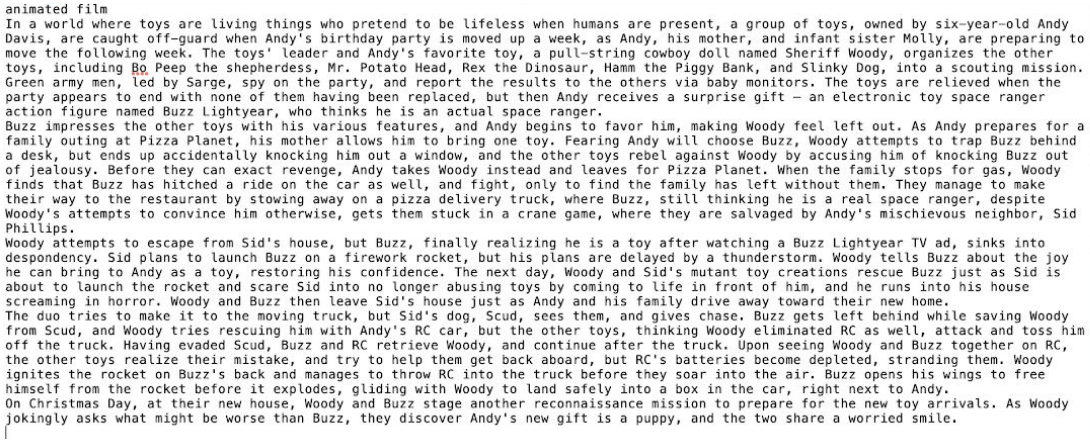

Figure 8: End result of pre-processing, corpus stored in updatedCorpus.zip

The actual running of LDA was performed by the MALLET topic model package developed by researchers at the University of Massachusetts - Amherst [6]. For our experimentation, we stored the output of running MALLET on 'updatedCorpus.zip' with 100 topics on July 24, 2020. This was then post-processed in order to build individual user recommendations by associating each item's topic model with its movieID and then taking advantage of each user's rating for a particular movie to build a user persona based on topics they rated highly (topics commonly found in movies they rated highly). This user persona was lastly used to generate a list of movie recommendations using collaborative filtering or neighborhood based recommendation to build a similarity matrix for each user in the neighborhood. The entire post-processing aspect of our implementation is contained in 'recommender.py' which outputs a list of a given length of movie titles in English as recommendations for given userID, and a given number of neighbors to determine how large the neighborhood of users should be.

For example, for the user with user_id 1 with a neighborhood size of 100 users, when queried for 100 recommendations, our algorithm produced the following:

['American Beauty (1999)', 'Jurassic Park (1993)', 'Shawshank Redemption, The (1994)', 'Star Wars: Episode VI - Return of the Jedi (1983)', 'Star Wars: Episode V - The Empire Strikes Back (1980)', 'Matrix, The (1999)', 'Gladiator (2000)', 'Shakespeare in Love (1998)', 'Raiders of the Lost Ark (1981)', 'Braveheart (1995)', 'Silence of the Lambs, The (1991)', 'Forrest Gump (1994)', 'Groundhog Day (1993)', 'Men in Black (1997)', 'Pulp Fiction (1994)', 'Terminator 2: Judgment Day (1991)', 'Mission: Impossible (1996)', 'X-Men (2000)', 'Fugitive, The (1993)', 'Being John Malkovich (1999)', 'L.A. Confidential (1997)', 'Indiana Jones and the Last Crusade (1989)', 'Ghostbusters (1984)', 'Galaxy Quest (1999)', 'Total Recall (1990)', 'As Good As It Gets (1997)', 'Good Will Hunting (1997)', 'Usual Suspects, The (1995)', 'Speed (1994)', 'Chicken Run (2000)', 'Patriot, The (2000)', 'Gone with the Wind (1939)', 'Star Wars: Episode I - The Phantom Menace (1999)', 'Babe (1995)', 'Stand by Me (1986)', 'When Harry Met Sally... (1989)', 'Willy Wonka and the Chocolate Factory (1971)', 'GoodFellas (1990)', 'Hunt for Red October, The (1990)', 'Independence Day (ID4) (1996)', 'Casablanca (1942)', 'Terminator, The (1984)', 'Rock, The (1996)', 'Face/Off (1997)', 'Beetlejuice (1988)', 'Godfather, The (1972)', 'Thelma & Louise (1991)', 'Twelve Monkeys (1995)', 'Breakfast Club, The (1985)', 'Who Framed Roger Rabbit?

(1988)', "There's Something About Mary (1998)", 'Sense and Sensibility (1995)', 'Dances with Wolves (1990)', 'Lion King, The (1994)', 'Star Trek: First Contact (1996)', 'True Lies (1994)', 'Fight Club (1999)', 'Graduate, The (1967)', 'Fifth Element, The (1997)', 'Seven (Se7en) (1995)', 'Amadeus (1984)', 'Clerks (1994)', 'Batman (1989)', 'Aliens (1986)', 'African Queen, The (1951)', 'Abyss, The (1989)', 'Godfather: Part II, The (1974)', 'Contact (1997)', 'Green Mile, The (1999)', 'Maverick (1994)', 'Alien (1979)', 'Austin Powers: The Spy Who Shagged Me (1999)', 'Star Trek: The Wrath of Khan (1982)', 'Jaws (1975)', 'Lethal Weapon (1987)', 'Indiana Jones and the Temple of Doom (1984)', 'Election (1999)', 'Superman (1978)', 'Romancing the Stone (1984)', 'Die Hard (1988)', 'Sling Blade (1996)', 'Raising Arizona (1987)', 'Last of the Mohicans, The (1992)', 'Few Good Men, A (1992)', 'Monty Python and the Holy Grail (1974)', 'Little Mermaid, The (1989)', 'My Cousin Vinny (1992)', 'Mission: Impossible 2 (2000)', 'Die Hard 2 (1990)', 'Perfect Storm, The (2000)', 'Exorcist, The (1973)', 'Rocky (1976)', 'Life Is Beautiful (La Vita bella) (1997)', 'Mask of Zorro, The (1998)', 'Star Trek: Generations (1994)', 'League of Their Own, A (1992)', 'Nightmare Before Christmas, The (1993)', 'Ghost (1990)', "Breakfast at Tiffany's (1961)", 'Bridge on the River Kwai, The (1957)']

Testing

Since the output of our implementation was simply a generated list of recommendations, we needed to find a control to compare our results to in order to evaluate its accuracy. To do so, we used Microsoft's production-level implementation of RBM, another probabilistic recommender model [2]. This algorithm was also tested on the MovieLens 1M dataset so it served as a good basis of comparison.

However, we had to limit the size of the neighborhood of users to build the comparisons with because the runtime of the similarity matrix portion of our algorithm being too slow. For initial comparisons of our data, we chose to make the size of the neighborhood 100.

To test the results of our implementation, in 'rbmcontrol.py', we computed the similarity percentage between our algorithm (after being ran with neighborhood size of 100 users for 100 recommendations) and the output of Microsoft's public RBM repository (once updated to produce the top 100 recommendations). To do so, we created a dictionary of the top 100 recommendations for each user, one dictionary for the output of Microsoft RBM and one for our LDA implementation.

Again looking specifically at the user with user_id 1, the only recommendation both algorithms produced for this individual was: ['Clerks (1994)'].

To generalize this process, we then calculated the number of entries found in each user's list in both models' dictionary and divided the total number of commonalities in both dictionaries by the total 100 recommendations produced. We then averaged this value for all users to find how similar our implementation of LDA's output was to Microsoft's implementation of RBM.

Results

After doing this, we found the similarity percentage to be 2.76%. For example, the only movie both algorithms found in common for user_id 1 was [‘Clerks (1994)']. While we had no baseline metric in mind, we presume that this disparity was because Microsoft’s implementation of RBM was able to use all of the users in the dataset for the final recommendation step, but our model only used the 100 users in the neighborhood for collaborative filtering due to the inefficient runtime of the similarity matrix building step in our implementation.

Reflection

Looking back at the objectives of our research, we were able to achieve a better understanding of probabilistic recommender systems through a literature review consisting of research papers, articles, videos, tutorials and direct testing of Microsoft’s RBM implementation. We also developed a limited proof of concept for the LDA model with room for future iterations. The primary next steps regarding our current proof of concept are its scalability and flexibility.

On the scalability aspect, due to the limited neighborhood size we had to use for our testing, addressing the runtime of the similarity matrix building would be the primary concern and is the biggest part of my partner’s current work with the lab. If we could perform collaborative filtering with all 6000 users instead of just 100, we would likely see an increase in accuracy (being measured as relative to Microsoft’s RBM model’s output). If we were able to address this concern, we could see whether or not our implementation is truly faulty, or if the low similarity value is simply from being tested on too small of a set of sample data.

Overcoming this obstacle would allow us to not only scale up our work and work on larger datasets, but also give way to generalizing the work and code we wrote. Right now, it pertains primarily to the MovieLens dataset but LDA is an algorithm that can be used on any textual based dataset. To take advantage of this, the preprocessing aspect would largely have to be generalized since now it uses many heuristics inherent to dealing with movie ratings to build MALLET’s input corpus but the post-processing largely just relies on creating user profiles and their similarity matrices and could largely stay the same.

References

[1] Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature.* 521:452–459. University of Cambridge, Cambridge, United Kingdom. 2015.
[2] M. González-Fierro et al. Microsoft. Examples - RBM MovieLens. GitHub repository. *Recommenders*. 2020.
[3] U. Gupta. DeepRecSys. Harvard University School of Engineering and Applied Sciences. Cambridge, MA, USA, 2020.
[4] X. He et al. Neural Collaborative Filtering. For the *26th International Conference on World Wide Web*. 2017.
[5] Q. Liu and D. Wang. Stein Variational Gradient Descent (SVGD): A General Purpose Bayesian Inference Algorithm. University of Texas - Austin. Austin, Texas. 2016.
[6] A. McCallum. MALLET. University of Massachusetts - Amherst. Amherst, Massachusetts, 2016.
[7] D. Ofer. Medtronic. Movies Data Clean. Kaggle dataset. Tel Aviv-Yafo, Israel, 2017.

[8] F. Ricci et al. Introduction to Recommender Systems Handbook. In *Recommender Systems Handbook*, pages 1-35, 2011. Springer Science and Business Media.
[9] J. Robischon. Movies Data Clean. Kaggle dataset. Parker, Colorado, USA. 2018.
[10] J. Rocca. Bayesian inference problem, MCMC and variational inference. Towards Data Science. 2019.
[11] A. Sharma. Restricted Boltzmann Machines — Simplified. Towards Data Science. 2018.
[12] H. Steck. Markov Random Fields for Collaborative Filtering. Netflix. Los Gatos, CA, USA, 2019.
[13] C. Wang and D. Blei. Collaborative Topic Modeling for Recommending Scientific Articles. Princeton University Department of Computer Science. Princeton, NJ, USA, 2017.
[14] J. Wilson. Improving Collaborative Filtering based Recommenders using Topic Modelling. Flytxt Mobile Solutions Research and Development Department. Trivandrum, India, 2014.
[15] J. Zhu. Probabilistic Machine Learning: Models, Algorithms and a Programming Library. Tsinghua University - Lab of Brain and Intelligence State Key Lab for Intell. Beijing, China. 2018.