A Survey of Recommendation Algorithms

Junye Lu

jl3946@cornell.edu

Snigdha Singhania

ss4224@cornell.edu

1 Introduction

1.1 Background

Television has been around since the early 20th century. At the very beginning, people could only watch programs offered by major networks. The choices available were scarce, but more recently, with the invention of internet, the entertainment industry was revolutionized where viewers now can choose from millions of shows, musics, movies and much more. How should one know what to watch? In the past, viewers could probably look for recommendations or promotions at different magazines, newspapers. They could also read reviews by online users and ask for more relevant movies on a online forum. Netflix simplified this sometimes complicated system by offering users a smarter, more personalized choices using recommendation systems.

1.2 Overview

According to Wikipedia [1], a recommendation system is "a subclass of information filtering system that seeks to predict 'rating' or 'preference' a user would give to an item". Currently, the application of recommendation system in technology scene is everywhere. For example, TicTok uses recommendation algorithms to analyze user preferences based on the time a user remains on a specific piece of content and pushes more related content to the user to increase engagement. Amazon analyzes the web footprints of users and comparing similarities between users to better promote relevant products to increase sales.

1.3 Types of Recommendation Systems

Recommender Systems are divided into 3 broad categories - Content-Based, Collaborative Filtering (CF) and Hybrid Recommender Systems [2].

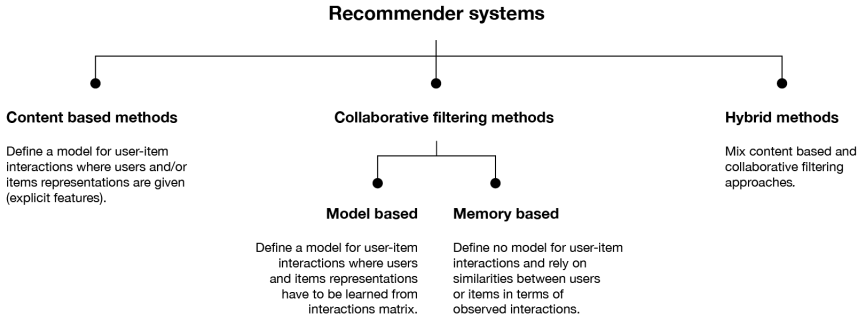

Figure 1: Summary of types of Recommender Systems [3]

Content-based systems attempt to find similar items based on attributes and recommends them to users based on their history of preference. Collaborative Filtering, on the other hand, relies on user and item interactions and disregards

item contents. It attempts to find similar users based on similar interactions and recommends items accordingly. Lastly, Hybrid recommender systems combine the two techniques mentioned previously, i.e., it leverages both item attributes and user interactions to suggest items. Figure 1 summarizes the different types of recommender systems discussed above.

In this paper, we will briefly survey different recommendation algorithms, and compare their effectiveness on the MovieLens-100K dataset.

2 Related Work

Early approaches to personalized recommendation systems focused primarily on content-based or collaborative filtering (CF)-based recommendation [4]. Shani and Gunawardana evaluated different recommendation systems by performing offline and online experiments on user behaviors [5]. They also talk about different ways to evaluate recommendation properties such as prediction accuracy, rating and ranking measures. Koren et al evaluate the impact of latent factors such as implicit feedback, temporal effects and confidence intervals on the accuracy of predictions [6]. Their results show that results improve as factor dimensionality increases. Lemire and Maclachlan introduce Slope One which uses an intuitive principle of "popularity differential" between items for users to predict movie ratings [7]. They group items in pairs, and compare how much one item is preferred as compared to the other.

Isinkaye et al. discuss the principles, techniques, and evaluation metrics of recommendation systems [8]. Said and Bellogin explain problems associated with comparing metrics across algorithms [9]. For fair comparisons, they control evaluation dimensions being benchmarked such as dataset, data splitting, evaluation strategies, and metrics.

We model our experiments based on techniques discussed in these papers.

3 Dataset

For the purpose of our research, we chose MovieLens 100K dataset [10] for our experiments. MovieLens dataset is one of the most comprehensive dataset on user preferences on movies. The dataset consists of 100,000 ratings from 943 users on 1682 movies. Every user has rated at least 20 movies. The ratings range from 1-5. The dataset also contains genre information for the movies and demographic user information. 17 genres are used to describe each movie in the dataset. These are Action, Adventure, Animation, Childrens, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War and Western.

Table 1: Dataset: MovieLess 100k

# Movies	# Users	# Ratings	Sparsity
1,682	943	100,000	93.695%

4 Approach

Recommendation systems are separated into two main categories: 1) Rating Algorithms and 2) Ranking Algorithms. Rating Algorithms predict ratings for unknown movies for users and ranking algorithms generate recommendations for users. We used some open-source Python libraries such as Surprise, Spotlight and LightFM for implementing our recommendation models.

4.1 Rating Algorithms

User Average Rating (Baseline) In order to compare performances among different rating algorithms, we use average user rating as the baseline for analysis. For this algorithm, the rating of the test movie is calculates as the mean of the same user's ratings from the training set.

***k*-Nearest Neighbors** *k*-NN is a non-parametric, lazy learning, collaborative filtering method. Here, the existing ratings of a user is used to compare with ratings of users in the training set. The resulting rating is the mean of the rating for the target movie in `top-k` nearest neighbors found in the training set. For our experiments, we create and evaluate user-based models.

Slope One Slope One is a simple, linear, item-based collaborative filtering algorithm that compares the average rating differences between the user and the training set, and adjust the rating of the target movies of the training set with this difference [7].

Co-Clustering Co-clustering is also a collaborative filtering algorithm that is both user-based and item-based similar to k-means. The algorithm clusters user and item at the same time to identify users with the similar preferences, and then return the mean ratings for the target movie of that particular co-cluster.

Explicit Feedback-based Matrix Factorisation Explicit feedback-based matrix factorization algorithm based on a classic matrix factorization approach, with latent vectors used to represent both users and items. Their dot product gives the predicted rating for a user-item pair [11].

4.2 Ranking Algorithms

Ranking algorithms differs from rating algorithms in that it offers ranking as a proxy to actual preference scores [12].

Most Popular (Baseline) We use most popular movies as baseline for comparison, given that intuitively the popularity of movies on a large audience is not independent. Thus, the top-50 movies in the training set are used to predict the items in the test set.

Markov Chain-based Sequential Models A Markov Chain calculates the outcome of a given event based the previous event [13]. Particularly beneficial when the data is sequential in nature, the likelihood of watching a given movie is determined based on the transition matrix built using 1^{st} order transitions in the training data.

Implicit Feedback-based Matrix Factorisation Implicit feedback-based matrix factorization algorithm similar to explicit feedback-based matrix factorization model, uses negative sampling where a lack of preference by the user as the implicit preferences) to give predicted rating for a user-item pair [11].

Implicit Feedback-based Matrix Factorisation with Item Attributes An example of a hybrid recommendation system, this algorithm includes item attributes as part of the matrix factorization [14]. Item features are particularly helpful in solving cold-start problems.

5 Experiments

5.1 Data Pre-processing

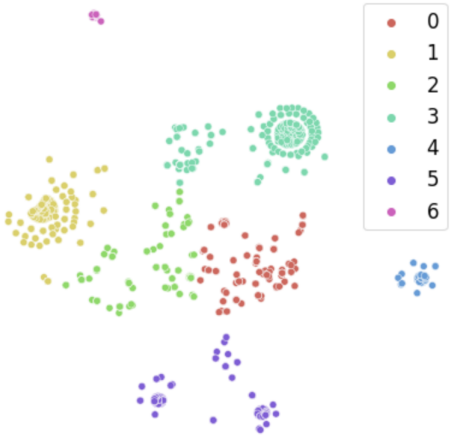

Figure 2: Visualizing Movies based on Genres using t-SNE

We pre-process the original dataset to remove movies that have <= 2 occurrences in user interactions. The sparsity after pre-processing reduces to 92.821%. Next, we split this dataset into 3 groups - training, validation and test. For this, we group the dataset by `userId` and sort it based on the timestamps. The latest movie rated by each user as per this

sequence forms the test set, while the last-but-one movie forms the validation set (Leave-One-Out Cross-Validation). The remaining data is used to train the model. Table 2 highlights the number of interactions across these splits.

Table 2: Dataset Partition: Interactions

Train	Validation	Test	Sparsity
97,837	943	943	92.821%

The validation set is used for hyper-parameter tuning. The best hyper-parameters are selected. The train and validation data is then combined and run on the test set to evaluate our models.

We visualize the *genres* of every movie, which we use as attributes for our attribute based models as shown in Figure 2. t-SNE is used to reduce the feature attributes from 17 to 2 dimensions. Next, *k*-Means is used to cluster and color-code the data. The figure shows us that there are some clearly defined clusters in the dataset which can boost the performance of learning algorithms, particularly due to problems owing to cold-start or short user sequences.

5.2 Evaluation

The following measures of accuracy are used to evaluate the performance of our recommendation models. We split them into 2 categories - based on the type of the models they evaluate.

5.2.1 Rating-related Measures

Mean Absolute Error Also known as MAE, it calculates the deviation of the predicted movie ratings from the actual ratings. If *N* is the number of test users,

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|predicted_i - actual_i|$$

Root Mean Square Error Similar to MAE, RMSE calculate the deviation from the actual ratings where the errors are squared. This measure places larger emphasis on outliers and errors as compared to MAE. If *N* is the number of test users,

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(predicted_i - actual_i)^2}$$

5.2.2 Ranking-related Measures

We use our recommendation models to generate *top-50* recommendations per user and use the following measure to evaluate their performance.

Hit Rate When the test-item for the given user occurs in the *top-k* items predicted by the model, it is a hit. The hit rate of the system is the ratio of the hit count to the total number of test users (*N*).

$$HitRate = \frac{\#Hits}{N} * 100\%$$

Normalized Discounted Cumulative Gain Also known as NDCG, this is a popular measure when the order of items predicted by the model is of relevance. For the *top-k* recommendations by the model, we approximate NDCG as follows (where *N* is the number of test users):

$$NDCG = \frac{1}{N}\sum_{i=1}^{N}\frac{log_{10}2}{log_{10}(rank_i + 2)}$$

5.3 Implementation (Code)

We make use of several open-source libraries to implement our experiments.

Spotlight An open-source library, which is built using PyTorch, to experiment with both deep and shallow recommender systems [11]. In our work, we use it to implement Explicit Feedback-based Matrix Factorization technique for rating the movies.

Surprise Another Python library, which is used primarily for building systems that require explicit rating of a dataset [15]. We use this library to implement *k*-NN, Slope One and Co-Clustering algorithms. The name stands for *Simple Python Recommendation System Engine*.

LightFM Not only does LightFM support both implicit and explicit recommendations, but can also be configured to incorporate user and item metadata into the factorization technique [14]. Thus, we use this open-source library for implementing both Implicit Feedback-based Factorization techniques, with and without item attributes.

While all the libraries have the MovieLens-100k dataset in-built, we use the original dataset to keep the train-test split consistent across all models, for fair comparison.

Additionally, **HyperOpt** is used for hyper-parameter optimization for the algorithms specified above [16].

Self We implement Markov-based model ourselves with some inspiration from Shani et al [13]. We also write our own baselines and evaluation measures.

All our code is available in this notebook.

5.4 Results

5.4.1 Rating Algorithms

Root Mean-Squared Error (RMSE) is used to evaluate the best hyper-parameters based on the validation set. Figures 3-5 shows the results for these optimisations, where the optimal result is highlighted in Red. The test dataset is then used to evaluate the model, the results of which are summarised in Table 3.

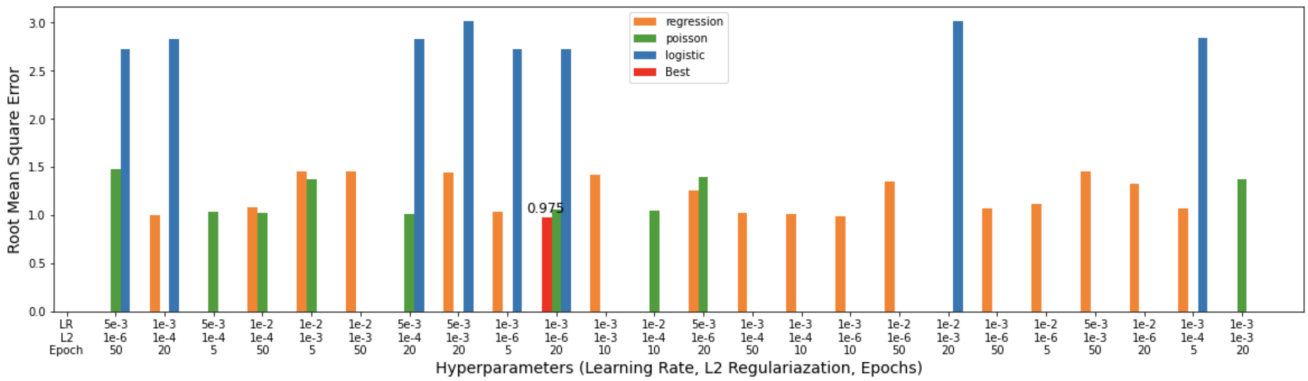

Figure 3: Hyperparameter Tuning for Explicit Factorization

Table 3: Rating Models Comparison on Test Dataset

Metric	Model Type				
	User Average Rating	Explicit Factorization	*k*-NN	Slope One	Co-Clustering
RMSE	1.1563	**1.0201**	1.0575	1.0277	1.0528
MAE	0.9185	**0.7886**	0.8349	0.7961	0.8140

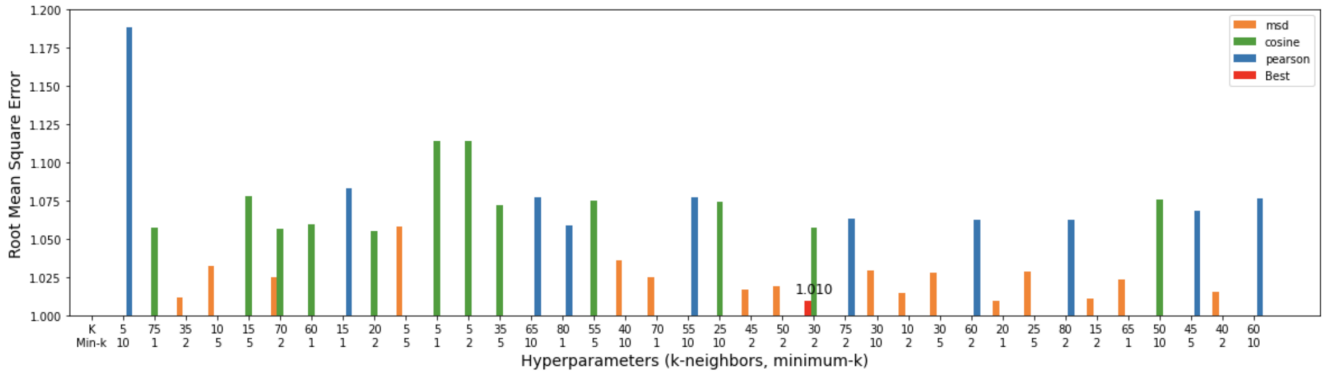

Figure 4: Hyperparameter Tuning for *k*-NN

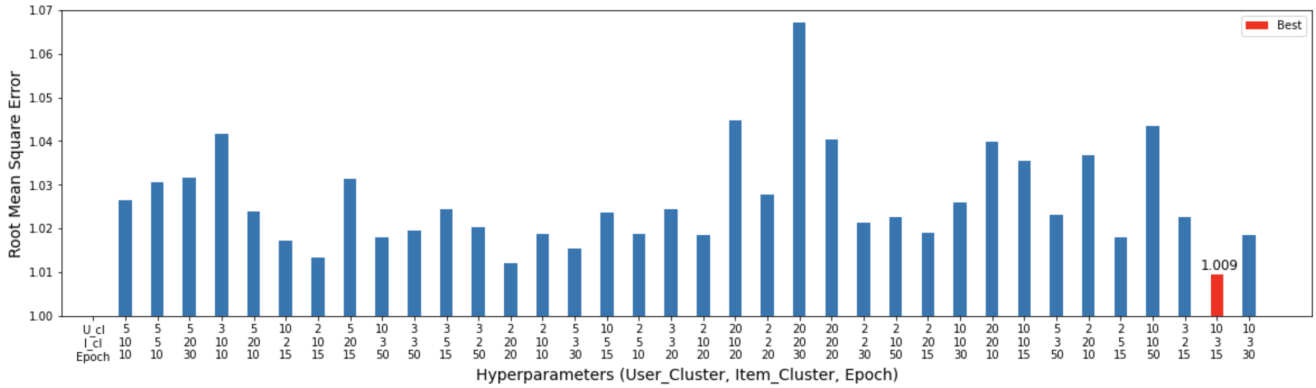

Figure 5: Hyperparameter Tuning for Co-Clustering

5.4.2 Ranking Algorithms

Hit Ratio is used to identify the best hyper-parameters for these algorithms which is evaluated on the validation set. Figures 6 and 7 highlight the results for these optimisations, where the optimal parameters are marked in Red. The test dataset is then used to evaluate the model using these hyper-parameters, the results of which are summarised in Table 4.

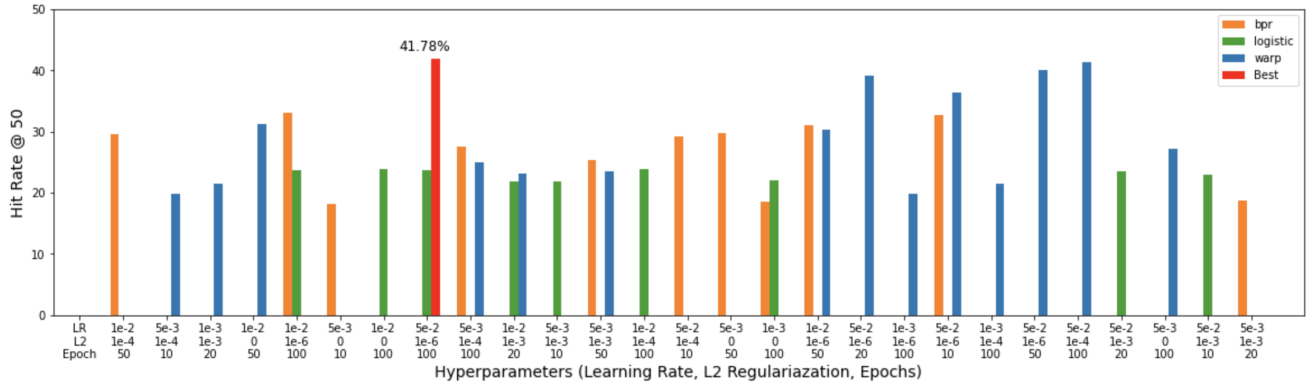

Figure 6: Hyperparameter Tuning for Implicit Factorization

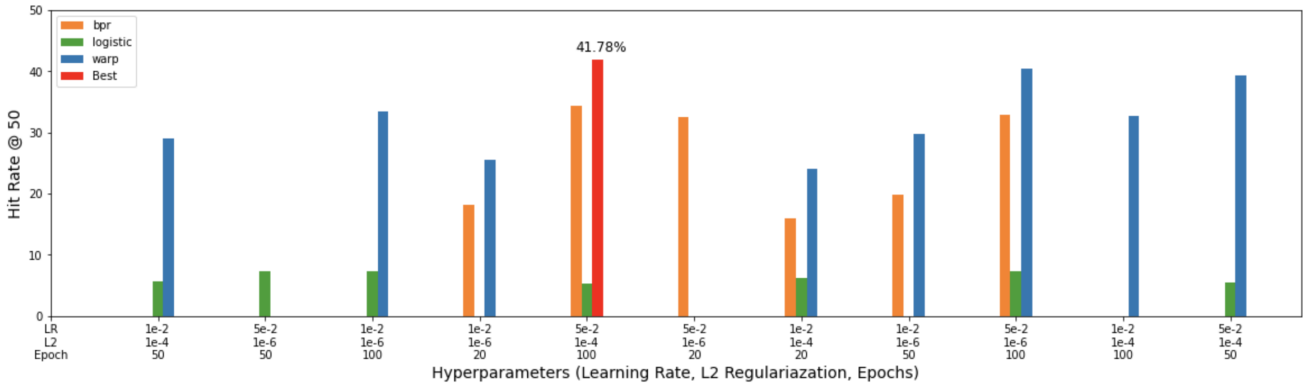

Figure 7: Hyperparameter Tuning for Implicit Factorization with Item Attributes

Table 4: Ranking Models Comparison on Test Dataset

Metric	**Model Type**			
	Most Popular	Implicit Factorization	Implicit Factorization (Attributes)	1st order Markov
Hit Rate (%)	15.27	35.7370	**36.3733**	29.9046
NDCG	0.0443	0.1006	0.1035	**0.1126**

5.5 Discussion

Rating Algorithms From the results seen in Table 3, while all models beat our User Average Baseline, *k*-Nearest Neighbor performs the worst of the experimental models. Poor performance of neighborhood-based techniques can be attributed to the high sparsity (92.821%) in the dataset as well as popularity bias which weakens the algorithm family. Figure 2 shows clustering based on human-generated genres. Computer generated latent factors can create factors in more dimensions and better classify the movies in our dataset, resulting in Explicit feedback-based Matrix Factorization producing the best results. Slop One, too, alleviates problems of sparsity and scalability which are significant drawbacks of neighborhood techniques, thereby producing results much closer to the factorization method.

Ranking Algorithms We see that the Implicit feedback-based models outperform Most Popular(Baseline) as well as the 1st order Markov model (Table 4). While we achieved best results for the Markov Chain-based Sequential model during the earlier deliverable, tuning the hyper-paramters for implicit models improved their results significantly. We also introduced a new loss function in the implicit models, WARP (Weighted Approximately Ranked Pairwise), which beat the loss functions used (BPR and Logistic) from the previous deliverable. Adding item attributes improve the results on the test set, although insignificantly. This shows that item attributes provided by the dataset are insufficient, which is also in accordance to the conclusion from Rating algorithms. Further, unsurprisingly, Markov-based models produces the highest NDCG score, as it evaluates the sequential efficacy of the model and is the only sequential model we evaluate.

6 Conclusion

In this paper, we look at algorithmic, explainable approaches for item recommendation. We look at several neighborhood-based and latent-factor approaches. In both the rating and ranking categories, latent models show the best results. In compliance with Koren et al, we see that the results improve with increasing feature dimensionality [6].

In the future, we can explore higher order Markov models and other more refined sequential-models to improve our understanding of the dataset. We can also look at effects of incorporating user attributes to our Hybrid model.

References

[1] "Recommender system." https://en.wikipedia.org/wiki/Recommender_system.

[2] R. Mu, "A survey of recommender systems based on deep learning," *IEEE Access*, vol. 6, pp. 69009–69022, 2018.

[3] B. Rocca, "Introduction to recommender systems." https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada, 2019.

[4] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in *Recommender systems handbook*, pp. 1–35, Springer, 2011.

[5] G. Shani and A. Gunawardana, *Evaluating Recommendation Systems*, pp. 257–297. Boston, MA: Springer US, 2011.

[6] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[7] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," in *Proceedings of the 2005 SIAM International Conference on Data Mining*, pp. 471–475, SIAM, 2005.

[8] F. Isinkaye, Y. Folajimi, and B. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 261 – 273, 2015.

[9] A. Said and A. Bellogín, "Comparative recommender system evaluation: benchmarking recommendation frameworks," in *Proceedings of the 8th ACM Conference on Recommender systems*, pp. 129–136, 2014.

[10] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[11] M. Kula, "Spotlight." https://github.com/maciejkula/spotlight, 2017.

[12] Y. Ji, A. Sun, J. Zhang, and C. Li, "A re-visit of the popularity baseline in recommender systems," *arXiv preprint arXiv:2005.13829*, 2020.

[13] G. Shani, D. Heckerman, and R. I. Brafman, "An mdp-based recommender system," *Journal of Machine Learning Research*, vol. 6, no. Sep, pp. 1265–1295, 2005.

[14] M. Kula, "Metadata embeddings for user and item cold-start recommendations," in *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015.* (T. Bogers and M. Koolen, eds.), vol. 1448 of *CEUR Workshop Proceedings*, pp. 14–21, CEUR-WS.org, 2015.

[15] N. Hug, "Surprise." https://github.com/NicolasHug/Surprise, 2015.

[16] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International conference on machine learning*, pp. 115–123, PMLR, 2013.