Chapter 5

Proximal Policy Optimization based Hybrid Recommender Systems for large scale recommendations

With substantial increase in digital information being accessed via the internet, recommender systems(RS) have grown in popularity. They help provide customized recommendations to users by selecting a small number of products from a large set of items. Today, RS have been applied to diverse domains such as in e-commerce, media, songs, news, advertisement, etc. Recommender systems are designed to predict and recommend items that users might be interested in based on their preferences and behavior. The relevance of items in recommender systems is typically determined by a combination of factors, and ranking plays a crucial role in presenting the most relevant items to users. "Top-N" recommendations in recommender systems refer to the practice of presenting a user with a list of the N most relevant items based on their preferences or historical interactions. The ranking of these items is crucial as it determines the order in which they are displayed to the user. Machine learning based approached including Reinforcement Learning methods have been extensively applied for recommender systems. With the increasing item-user space, scalability remains a key challenge for recommender systems. However, most ex-

isting policy gradient methods for recommender systems suffer from high variance leading to an increase in instability during the learning phase. Policy Gradient approaches such as PPO have shown to be effective in large action spaces (a large number of items) as they learn the optimal policy directly from the samples. We utilize the PPO algorithm for traiinng the RL agent modeling the collaborative filtering process as an MDP. We also address the cold start problem in Collaborative filtering using autoencoder-based content filtering. Proximal Policy Optimization algorithm is considered amongst the most effective reinforcement learning methods presently, delivering state-of-the-art performance and even outperforming Deep Q learning approaches. In this work, we propose a switching hybrid recommender system that combines two separate recommender system approaches and provide the top N list of items to the user. A switching hybrid system can switch between recommender systems approaches based on some criterion and can compensate for the shortcomings of one constituent recommender system by employing the other counterpart in a given situation.

PPO utilizes the actor-critic framework and thus mitigates the high variance in Policy Gradient Algorithms. Further, we address the cold start issue in Collaborative filtering with autoencoder-based content filtering. Proximal Policy Optimization (PPO) methods are today considered among the most effective reinforcement learning methods, achieving state-of-the-art performance and even outperforming Deep Q learning methods.

Recently, recommender systems based on Reinforcement Learning have been proposed and shown to be effective for recommendation task [132, 31, 42, 58, 95, 91, 85, 133]. Multi-Armed Bandits based methods have been proposed for recommender systems [52] that learn the user's preferences through constant interaction. MAB methods, on the other hand, presume that user preferences remain static and do not vary over time during the recommendation process, and hence fail to capture dynamic user preferences [37]. In [85], the authors proposed an MDP-based recommender system and utilized Q-learning for the training. However, Q-learning

becomes intractable with a very large number of items as it is lesser efficient in very large action space. Model-free RL approaches have been applied to recommender systems and are generalized as-value based [133, 8, 19, 137] and policy-based methods [31, 58]. Value-based methods require computing the Q values of all the actions for a state and then selecting the action with the highest Q-value as the best action for that state. As a result, calculating Q-values for all actions for each state can be inefficient when action space is very large, i.e. millions of items like in recommender systems problems [22]. Policy based methods are thus preferred if the action space is large. Model-free methods calculate estimates using either the Monte Carlo or the TD Learning methods. Monte-carlo methods deals with high variance in large scale tasks, whereas TD approaches provide better efficiency through bootstrapping techniques, however they do suffer from Deadly triad [102] problem, that arises due to the combination of function approximation, bootstrapping, and offline training and leads to instability and inefficiency.

Monte-Carlo approaches can result in an extremely large action space and an unbounded importance weight of training samples, giving rise to instability and slower convergence [138]. An RL-based recommender system was also proposed to address the cold-start problem in recommender systems [137], where the authors utilized a Q-leaning-based strategy for learning. Deep Q Network (DQN) exploits a neural network for functional approximation and for stability during training allowing it to learn a high-dimensional state space. DQN-based approaches have been proposed that combine the strong approximation capabilities of neural networks with Reinforcement learning [132, 134, 19]. However, DQN cannot be employed when the action space is continuous as it is necessary to select actions that maximize the current action value in each state. Thus, DQN-based approaches are better suited to small discrete action spaces, whereas recommender systems typically contain vast and high-dimensional action spaces. Therefore, DQN-based approaches are mostly suitable for small discrete action spaces, whereas recommender systems typically contain vast and high-dimensional action spaces [19]. Policy-based tech-

niques prove useful for dealing with huge action spaces, such as continuous spaces with an unlimited number of actions [19]. In policy-based techniques, the agent learns the policy directly and selects an action from a probability distribution of the action space. Furthermore, policy-based approaches can learn stochastic policy, unlike value- based approaches, and therefore handle the exploration/exploitation conflict automatically. However, the conventional Policy Gradient technique has significant variance due to gradient estimation, and a large state space and action space would result in sample inefficiency [92, 113]. Proximal Policy Optimisation is an Actor-Critica method that incorporates the benefits of both DQN and Policy Gradient-based approaches and has achieved state-of-the-art performance in Reinforcement Learning, outperforming even Deep Q learning based methods. With a clipped objective surrogate function, PPO ensures minimal variance during learning by ensuring that the updated policy isn't too different from the old policy. The reduction of variation contributes to the increased stability of the learning process. Furthermore, the PPO agent limits the policy gradient step and decreases the variance of the estimation using an advantage function so that it does not deviate too far from the initial policy, resulting in excessively large updates that frequently make the policy unstable. We address the challenges described above utilizing the state-of-the-art method Proximal Policy Optimization, which uses on poilcy learning and therefore avoids the deadly triad problem. We also employ the Probabilistic Matrix Factorization method to derive our state from the user and item matrices. This strategy further addresees the sparsity problem as the PMF approach performs well on big, sparse, and imbalanced matrices [65].

5.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [84], is a recently proposed and effective policy gradient algorithm and builds upon the concepts from the Trust Region Optimization Method(TRPO). TRPO introduced the idea of using trust regions to constrain policy updates, ensuring that updates do not deviate too far from the previous policy.

This stability is crucial for effective training. TRPO uses a second-order derivative matrix which increases the complexity for large-scale problems. PPO builds on TRPOs concepts but simplifies the implementation. It uses a clipped objective function that limits how much the policy can change in a single update, effectively creating a trust region without the computational complexity of TRPO. This allows PPO to maintain stability and improve sample efficiency while being easier to implement and tune.

Policy gradient methods such as Natural Policy Gradients, TRPO, etc involves a second-order derivative which makes it inefficient for large-scale problems as the computational complexity is too high for real tasks. PPO build on TRPO but instead uses a first-order optimizer like the Gradient Descent method, which makes it more applicable for large-scale tasks and considerably efficient for computation and can be used in both discrete and continuous environments. PPO tries to limit the difference from one policy to the next so as to regulate new policy not to be too different from the current one. PPO uses the Minorize-Maximization MM algorithm by iteratively maximizing a lower bound function M approximating the expected reward η locally (See Equation 2). PPO starts with an initial policy guess and finds a lower bound M for η at this policy. It optimizes M and uses the optimal policy for M as the next guess. It approximates a new lower bound again and repeats the iterations until the policy converges. There are two primary variants of PPO: PPO-Penalty and PPO-Clip. PPO-Penalty uses KL-divergence to measure how much policy changes in each iteration. KL-divergence measures the difference between two data distributions, p, and q.

$$D_{KL}(P \,||\, Q) = \mathbb{E}_x \log \frac{P(x)}{Q(x)}, M = L(\theta) - C.\overline{KL} \tag{5.1}$$

Here M is a lower bound funcion as described above, approximating expected reward. $L(\theta)$ equals $\mathbb{E}_t[\,\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{\mathcal{A}}_t$, denoting the expected advantage function($\mathcal{A}$) for the current policy which is estimated by the new policy and then reordered using the

probability ratio between the current and the old policy. The second term denotes the KL Divergence, which measures the difference between the two policies. The main objective in PPO-Penalty can be summarized as :

$$\max_{\theta} \text{ s.t. } \quad \mathbb{E}_t[\,\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{\mathcal{A}}_t\,] = \mathbb{E}_t[\,r_t(\theta)\hat{\mathcal{A}}_t] - \beta\,\mathbb{E}_t[\,\mathbf{KL}[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]] \tag{5.2}$$

, here β controls the weight of the penalty. It penalizes the objective if the new policy is different from the old policy and thus changes between iterations to ensure the KL Divergence constraint is satisfied. $\mathbb{E}_t[\,\mathbf{KL}[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]]$ is the KL-divergence between the old and the new policy. If it is higher than a target value, we reduce β and vice versa. PPO-Clip doesnt have a KL-divergence term in the objective and doesnt have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy. It uses a clipping function to make sure the new policy does not deviate much from the older policy. With clipped objective, we compute a ratio between the new policy and the old policy, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. This ratio measures how difference between two policies. If the new policy is far away from the old policy , then we clip the estimated advantage function. The objective function in PPO Clip can be summarized as:

$$L_\theta^{CLIP} = \mathbb{E}[\,\mathbf{min}(r_t(\theta)\hat{\mathcal{A}}_t, \mathbf{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{\mathcal{A}}_t\,] \tag{5.3}$$

Thus, PPO maintains the low variance as the new policy obtained from the old policy does not differ much, which is ensured through the clipped objective surrogate function. The advantage function above reduces the variance of the estimation, so it does not move too much away from the original policy, avoiding large up- dates which can lead to unstable policy. Therefore, if the probability ratio between the new policy and the old policy falls outside the range $(1-\epsilon)$ and $(1+\epsilon)$, the advantage function will be clipped. For eg, if ϵ is .25, then the $r_t(\theta)$ could vary between .75 to 1.25. And finally, it takes the minimum of the two, i.e., clipped and unclipped objective, and so thus, the algorithm doesn't become too greedy and

prohibits making too many updates at once.

5.2 Problem statement

In Reinforcement learning problem, the agent selects the appropriate action based on it's current state. An MDP can be used to model the recommender system. The RL agent interacts with the environment and obtains the feedback. Next, the agent transitions to the next state. To obtain the latent vector representing the state, we employed probabilistic matrix factorization.

The recommender system provides an item to the user and subsequently obtains feedback on the item. After taking into account the observed rating, the recommender updates its information about the user and provides a new item to user at the next step. Assume that this recommender-user process continues for T time steps. The recommender system's aim is to provide the most interesting items to the user while maximizing the reward obtained over T steps. In this work, we addressed the cold start problem using a switching Hybrid recommender setting. We categorised users as either cold or hot based on the amount of ratings they provided. Furthermore, we used autoencoder-based content filtering for users who had supplied relatively few ratings, i.e., cold set users, and MDP-based collaborative filtering for the other group of hot users. Further, we used demographic information for recommendation for the users who have provided no ratings at all.

5.3 MDP based Collaborative filtering

We modelled the collaborative filtering process as an MDP in our work.

State Space S: The state $s_t^1, ...s_t^n \in S$ is denoted as a latent feature vector of the item with meta information. We obtain the state from the Matrix factorization technique. Matrix factorization is a collaborative filtering approach used in recommender systems that works by breaking down the user-item interaction matrix

into two lower dimensionality rectangular matrices. To obtain our MDP states, we employed Probabilistic Matrix factorization approach [122].

Action space A: An action $a_t^1, ...a_t^n \in A$, is to recommend items to a user at time t based on the current state s_t. We recommend one item at a time to the user. The current policy decides the specific action to be selected. Action determines the next item to recommend. We utilize an actor policy network for predicting the rating. A policy $\pi : S \rightarrow A$, is a function that indicates for every state $s \in S$, the action selected by the agent in that state. In a standard actor–critic framework, the policy is specified by a parameterized actor function $\pi_{(\theta)} : S \rightarrow$ A. We use the actor-critic technique to implement our policy. The actor policy network is used to generate actions, while the critic is used to refine the actor's choices. We employ a policy network to predict the ratings of a specific item, and if the predict rating is 3 or above, we recommend the item most similiar to the current state. We utilised the nearest neighbour algorithm with cosine similarity to obtain similar items. Otherwise, we recommend a random item, therefore enhancing exploration. Next, the agent transitions to the next state (item). Multi-layer perceptron networks are used to implement both the actor and critic techniques. Further, to train our policy, we employed the PPO algorithm.

Transition: The transition between states is deterministic in our setting as the agent moves to the next state. The specific action is determined by the actor-critic framework and is dependent on the Agent's current policy.

Reward R: After the agent selects the action a_t at the state s_t, i.e., recommending the item to a user, the user provides his feedback. Our reward $r(s_t, a_t)$, is in the range [-1, 0 ,+1], based on the user feedback. We provide a -1 reward if the user's rating is less than or equal to 2. We give a reward of 0 if the user does not rate the item or gives a rating of 3. Similarly, a reward of +1 is granted if the user gives a rating of 4 or 5.

Discount factor $\gamma : \gamma \in [0, 1]$ determines how the agents values rewards in the

distant future relative to the immediate rewards. Specifically, when $\gamma = 0$, the agent only considers the immediate reward. In other words, when $\gamma = 1$, all future rewards can be counted fully into that of the present action.

5.4 Autoencoder based content recommender system

Autoencoders are neural networks that are utilised in unsupervised learning approaches such as generative modelling, dimensionality reduction, and so on. They encode the data into a reduced lower-dimensional form and recover the data from the reduced representation.

Content-based filtering approaches utilize product or item features such as tags, text description, reviews, and other meta information to provide recommendations. Content-based methods are more robust against the cold start issue than collaborative systems as they can utilize item/product information.

Embeddings are a form of word representation in natural language processing that allows words with similar meanings to have a similiar representation by mapping them to a latent vector space and are frequently created using neural networks. Word embeddings outperform one-hot encodings as they can retain similarity information between various words, that is lost in one-hot encodings due to their orthogonal nature.

We utilize neural network-based embeddings for the set of cold users, that is learned from textual information describing the items and demographic data linked with users. We used cosine similarity to determine the degree of similiarity between the movies. It is determined by the dot product of two vectors split by their magnitudes and ranges from -1 to 1. If two embedding vectors point in the same direction, they have a high cosine similarity score. Item embeddings can thus be derived from the content-related information linked with data.

Movielens' collection comprises user-generated movie tag information. Tags are

typically a single word or phrase. The document corpus is built up from movies and the tags associated with them. Each document, in particular, is the movie, along with all of its associated tags. Further, the tag documents in the corpus were used to generate the Term Frequency-Inverse Document Frequency (TF-IDF) representation. The TF-IDF technique assesses the relevance of a word in a corpus of documents. It is calculated by multiplying two metrics together: the number of times a word appears in a document (TF) and the inverse document frequency of the term across a group of documents (IDF). The greater the TF*IDF score, the rarer and hence more relevant the phrase, and vice versa. However, TF-IDF representations might be high dimensional, thus we utilise autoencoders to learn the representation of higher dimensional data into a matching lower form by minimising noise in order to compress the high-dimension TF-IDF vector data into low-dimension embeddings. In the ML 100K and Movielens 1m datasets, we further lowered the dimensions to 50 and 100, respectively. We use the cosine similarity metric to determine the similarity between the movies to obtain the similiar TF-IDF vectors, i.e., movies, corresponding to the movies rated highest by the present user. It computes the cosine angle between two vectors in high dimensional space and ranges from -1 to 1.

$$sim(A, B) = cos(\theta) = \frac{A.B}{||A||||B||}$$

5.5 MDP based PPO Algorithm

Due to the lack of user interaction history linked with the items, collaborative filtering algorithms often do not operate well with cold users. Therefore, for the set of cold set users, we utilise autoencoder-based content recommender systems, and for the other user set, we employ Reinforcement learning-based collaborative filtering. We divide the user sets into two in our solution. We used content-based filtering for the users in the cold set and an MDP-based recommender for the rest. Figure reff2 depicts the flowchart for the entire process. If the user is identified as a hot user

based from the interaction matrix rating profile, we apply MDP-based collaborative filtering, as shown in the Figure. It first obtains the state from PMF technique by utilising the user and item matrices. The state is then input into the RL policy, which is the PPO agent, that produces an action. The action is equivalent to recommending a product/item. Following that, the agent obtains a reward from the environment and moves on to the next state. For the complete cold start scenario, i.e., if a user has not provided any rating, we utilized the user embedding to search similar users and recommend the items corresponding to them. We used the demographic information included in the dataset to obtain user embeddings. We obtained user embeddings using the demographic information available in the dataset.

If the user has only given a few ratings, i.e., an incomplete cold start problem, we apply the above-mentioned technique with autoencoders to make recommendations to the user. Algorithms 4 and 5 represent the two algorithms for MDP-based collaborative filtering. The procedure of generating MDP transitions is depicted in Alg. 4. It first uses PMF to obtain the state, and then chooses the action based on the policy. The item is then recommended when the rating is obtained from the policy network. Finally, the agent receives the reward and moves to the next state. First, we gather the set of partial trajectories from Algorithm 4. The advantage function is then used to get the advantage estimate. Algorithm 5 depicts the training of the PPO agent.

5.6 Experiments

5.6.1 Experimental Settings

For our experiments, we used the Movielens datasets ML 1m and ML 100k. ML 1m has 1,000,209 anonymous ratings of 3,952 films submitted by 6,040 MovieLens users on a range of 1-5, whilst ML 100k contains 100000 evaluations of 943 individuals and 1682 films on the same scale. We utilised 80% of the dataset for training and 20% for testing the recommendations. We used grid search to fine-tune the hyperparameters.

Algorithm 4 Generating Transitions

s_0 initial state , obtained with pmf , described above
for $t = 1, 2, \ldots, N$ do
    Obtain the next state S_i for item I_i according to PMF
    select action according to policy
    obtain rating $r = \pi(s_i)$ from the policy network
    recommend item I_j as described in 5.1
    Get feedback from the user for item I_j as U_{rat}
    if $U_{rat} \geq 4$ then
        $reward = +1$
    else if $U_{rat} = 3$ then
        $reward = 0$
    else
        $reward = -1$
    end if
end for

Algorithm 5 MDP Based PPO Algorithm

Initial policy parameters θ_0
s_0 initial state , obtained with pmf , described above
for $k = 1, 2, \ldots$ do
    collect set of partial trajectories D_k on policy $\pi_{(k)} = \pi_{(\theta_k)}$ as in Algo. 1
    Estimate adavantage A using A(S,A) (Section 3.3)
    update policy by maximizing ppo clip objective using equation 4
    Compute policy update

$$\theta_{k+1} = \underset{\theta}{\text{argmax}}\, L^{CLIP}_{\theta_k}(\theta)$$

    using Gradient Descent where ,

$$L^{CLIP}_{\theta_k} = \mathbb{E}_{\tau \sim \pi_k}[\sum_{t=0}^{r} [\min(r_t(\theta)\mathcal{A}_t{}^{\pi_k}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\mathcal{A}^{\pi_k}]]$$

end for

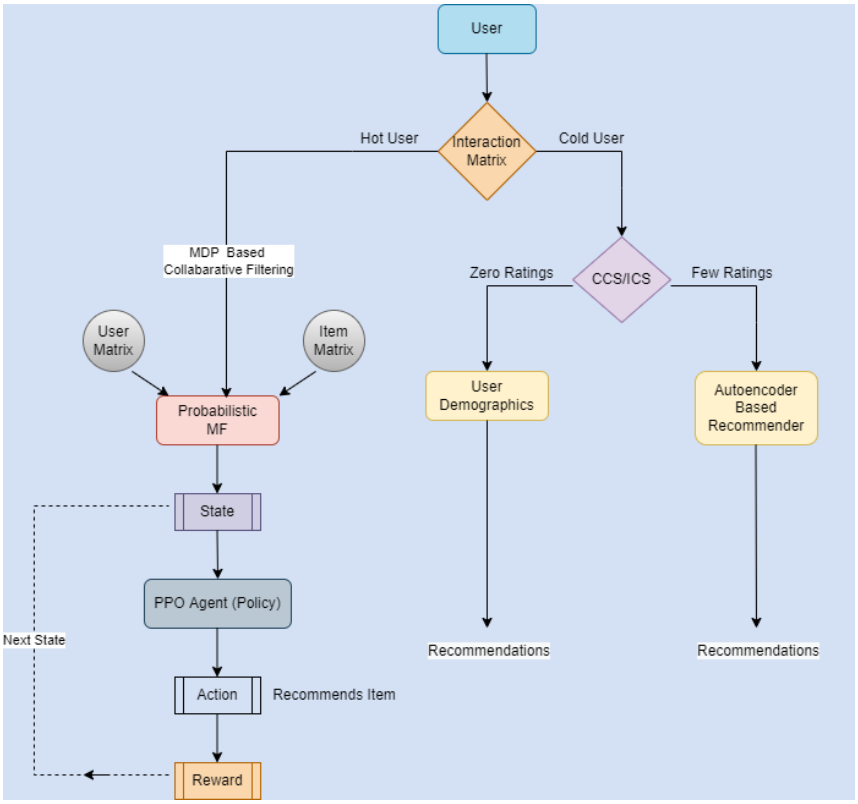


Figure 5.1: Flowchart depicting the modeling of recommendation process

We utilised a batch size of 128, and the learning rate for actor-critic networks was set to 0.001. We ran the PPO-based agent for over 1000 episodes. We varied the percent of the cold user, i.e., the least n percentage of users who have given a minimum number of ratings to the movies. The different values for clipping ratio, cold user threshold, and discount factor for the hyperparameters are also shown in Table 5.1. We also used different embedding sizes obtained from the PMF for finding the optimal ones on the two datasets. We used the Open AI Gym Environment to create a Markov Decision Process agent [11]. Further, we used the PPO stable-baselines framework for training the agent [78]. We ran the tests 5 times for the agent and reported the average results in the tables and figures below.

5.7 Evaluation baselines and criterion

For evaluating the recommender system approach, we used the popular metrics listed below.

Precision : It estimates the fraction of recommended items that are relevant to the user in a recommender system.

Precision@k = (# of recommended items @k that are relevant) / (# of recommended items @k)

Recall : It denotes the fraction of relevant documents that have been selected.

Recall@k = (# of recommended items @k that are relevant) / (total # of relevant items).

We used the following baselines for comaprisons:

Random : This approach randomly recommends the items to a user.

PopRank : A popularity based approach that recommends the most popular items.

Content Based Filtering(CBF) : CBF utilize item/product features to recommend other items similar to what the user likes, based on their past actions or feedback.

LinUCB : It's multi armed bandit based method that recommends items to the user depending on the contextual information about the user and items [51].

SVD : It is a popular algorithm utilizing Singular Value Decomposition for the process of recommendation [75].

Deep-MF : A state-of-the-art neural network architecture based Matrix Factorization approach for recommender systems [118].

Neural-MF : A state-of-the-art approach that utilize deep neural network architecture for Collaborative Filtering [40].

For SVD method, we utilised the same learning rate of 10^{-4}, regularisation rate of 0.02, and factor-size of 50 for ML-100K and 100 for ML-1M as our algorithm. The only hyperparameter in LinUCB method is α, which defines the balance between exploration and exploitation. We experimented with numerous values of α [0.1, 0.5, 1, 2] to find the best value. We used tf-idf and cosine similarity to extract the K most comparable movies from the dataset for content-based filtering. We

varied the value of parameter K to 20, 50, and 100. For Neural-MF method, we randomly initialized model parameters with a Gaussian distribution, with mean = 0 and standard deviation = 0.01, and used mini batch Adam for optimizing the model. For testing, we used a batch size of 512 and a learning rate of [0.0001, 0.0005, 0.001, 0.005]. Furthermore, for the neural network, we used three hidden layers for MLP. For Deep MF method, we set the depth of hidden layers to 3, the batch size to 256 and the varied the learning rate same as for NMF. We performed the experiments on Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz processor with 264 GB RAM. The system has x86_64 architecture with 40 CPUs and a multiple level cache. The experiment was carried out using the Python 3.6 framework on Ubuntu 16.04 LTS.

Table 5.1: Hyperparameters

Cold user percent	3%	5%	7%	10%
Discount Rate	0.85	0.90	0.95	0.99
Clipping Ratio	0.05	0.10	0.15	0.20

5.8 Results

In this section, we compare our method to the previously mentioned baseline methods for the evaluation metrics specified above. Our method is denoted as MDP below. Tables 5.2 and 5.3 show the results of comparing our proposed method to the baselines in terms of Precision@k and Recall@k on the two datasets. Our method outperformed the different baseline methods for most metrics on Movielens 100k dataset, except for the P@20 and R@1. For both of these metrics, Neural MF(NMF) gives the best performance.

Between the baselines, NMF performs the best, followed by the Deep CF algorithm and SVD, whereas PopRank method performs significantly better on Movielens 1m dataset as compared to Movielens 100k. LinUCB outperforms SVD on the ML-1M dataset for the recall@20 metric and on the P@20 on the Movielens-100k Dataset. Further, On Movielens 1m, our method outperforms the baseline by 3.10%, 3.86% and 6.58% in terms of P@1, P@10 and P@20, respectively, and 7.5% and 9.19% in

Table 5.2: Results for different metrics on Movielens 100k

Method	P@1	R@1	P@5	R@5	P@10	R@10	P@20	R@20
Random	0.1923	0.0711	0.1771	0.0816	0.1641	0.0782	0.1124	0.1271
PopRank	0.2421	0.1018	0.2291	0.1271	0.2163	0.1002	0.1572	0.1714
CB	0.2771	0.0818	0.2502	0.1014	0.2458	0.1072	0.1788	0.1702
LinUCB	0.3559	0.1112	0.3247	0.1271	0.3210	0.1354	0.2860	0.2035
SVD	0.3602	0.1066	0.3228	0.1271	0.3432	0.1613	0.2824	0.2453
DeepMF	0.3673	0.1042	0.3281	0.1271	0.3224	0.1428	0.3077	0.2283
NMF	0.3804	0.1187	0.3590	0.3455	0.3441	0.1601	0.3142	0.2438
MDP	0.3983	0.1154	0.3772	0.3661	0.3573	0.1676	0.3130	0.2512

Table 5.3: Results for different metrics on Movielens 1M

Method	P@1	R@1	P@5	R@5	P@10	R@10	P@20	R@20
Random	0.1611	0.0549	0.1422	0.0715	0.1380	0.0723	0.0950	0.1134
PopRank	0.3375	0.0774	0.2825	0.1007	0.2841	0.1121	0.2108	0.1823
CB	0.3110	0.1064	0.2902	0.1253	0.2712	0.1128	0.2047	0.1774
LinUCB	0.3449	0.1004	0.3211	0.1135	0.2988	0.1167	0.2213	0.2385
SVD	0.3505	0.0958	0.3441	0.1119	0.3167	0.1318	0.2762	0.2308
DeepMF	0.3588	0.1087	0.3256	0.1308	0.3043	0.1286	0.2551	0.2391
NMF	0.3611	0.1106	0.3572	0.1339	0.3180	0.1348	0.2714	0.2612
MDP	0.3723	0.1189	0.3416	0.1221	0.3314	0.1472	0.2944	0.2576

terms of R@1 and R@10, respectively. LinUCB method outperforms SVD algorithm over the ML-1M dataset for the recall@20 and on the P@20 metric on the Movielens-100k Dataset.LinUCB surpasses SVD for the recall@20 metric on the ML-1M dataset and for the P@20 metric on the Movielens-100k dataset. Furthermore, on Movielens 1m, our approach outperforms the baseline methods by 3.86% in terms of P@10 and 6.58% in terms of P@20, and 9.19% in terms of R@10. Our method outperforms the baseline methods by 4.70%, 5.06% and 4.10% for the P@1, P@5 P@10 measures and 5.90%, 3.90% and 2.40% in terms of R@1, R@10 and R@20 for the Movielens 100k dataset, respectively. The effect of modifying different parameters such as the discount factor and clipping ratio on the Ml-100k dataset is further illustrated in Figures 5.3 and 5.4, respectively. Figure 5.5 demonstrates the impact of various embedding sizes produced from the PMF on the two datasets. Figure 5.6 shows the change in precision measure for the two datasets with regard to the cold user percentage threshold.

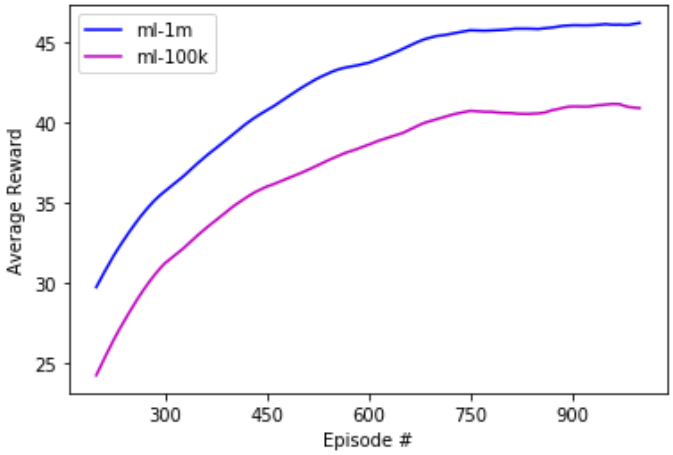
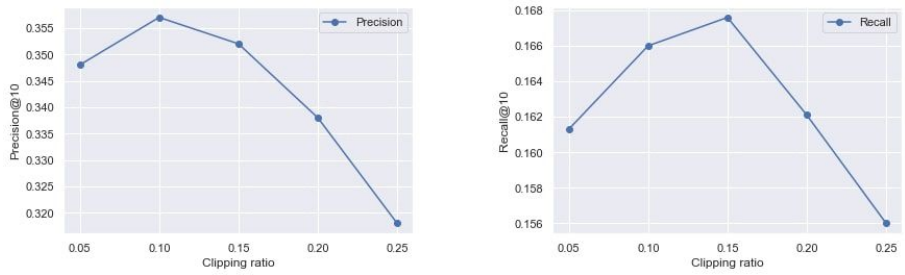


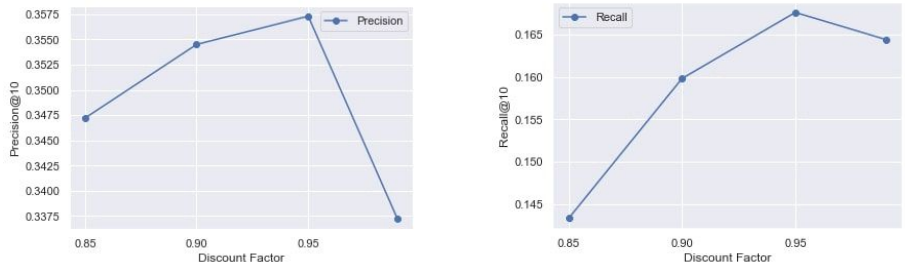
Figure 5.2: Reward Distribution of MDP over 2 datasets

We can observe that our method performs better on the Movielens 1m dataset in comparison to the Movielens 100k dataset. We could speculate that that because the Movielens 1m data has significantly more interactive rating data than the Movielens-100k data, we are able to learn a better model because the policy gradient algorithm has considerably more samples and can learn a better model with gradient descent. Figure 5.2 depicts our algorithm's reward distribution across two datasets, Movielens 100k and Movielens 1m. We can see that the algorithm converges faster on the Movielens 1m dataset than on the Movielens 100k dataset. We can observe stability in the reward distribution after 700 episodes in the Movielens 1m dataset, which we may attribute to the Policy gradient approach obtaining stability in convergence due to more sample data associated with the dataset than in the Movielens 100k dataset. Furthermore, as the number of episodes increases, the agent performs more stable learning on both datasets, signifying gradual convergence of the gradient descent method. Thus, we can observe the stability of the algorithm from the figure above. Another inference we can draw from the results is that the two deep learning methods perform substantially better on the ML 1M dataset than on the ML 100K dataset, as deep neural networks can learn a more complex model on the ML 1M due to availability more rating data. The drop in precision is steeper after the embedding size 50 for the Movielens 100k dataset, however for the Movielens 1m dataset, the variations in precision are more constant for different embedding sizes.



(a) (b)

Figure 5.3: Parameter Sensitivity by varying Clipping ratio



(a) (b)

Figure 5.4: Result of Varying Discount factor

As shown in Figure 5.6, the ideal threshold for the Movielens 100k dataset is 5%, whereas the optimal threshold for the Movielens 1m dataset is 7%. Precision, in particular, declines dramatically for both datasets as the percentage of cold users increases. Further, we can observe that Precision reduces significantly for both datasets as the percentage of cold users increases. We attain the maximum precision with a clipping ratio of 0.9 and a discount factor of 95%, as shown in Figures 5.3 and 5.4.

We utilised the Wilcoxon-Signed test [112] to determine the statistical significance of the results. It is a non-parametric test that, unlike the student t-test, does not assume a normal distribution over the input. The null hypothesis states that the two distributions do not differ statistically. The test was run at a 5% significance level utilising the MDP algorithm results against each of the baselines. The p-values obtained by the test are listed in Tables 5.4 and 5.5, with p-values less than 0.05 demonstrating rejection of the null hypothesis, implying a significant difference at a level of 5%. The p-values in the tables demonstrate that the improvement

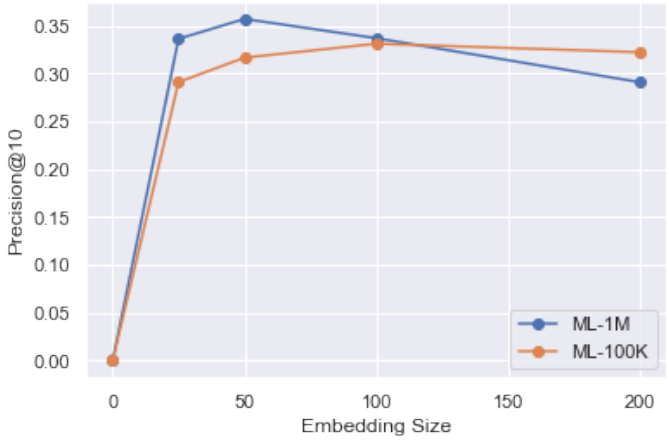


Figure 5.5: Effect of different embedding sizes on Precision

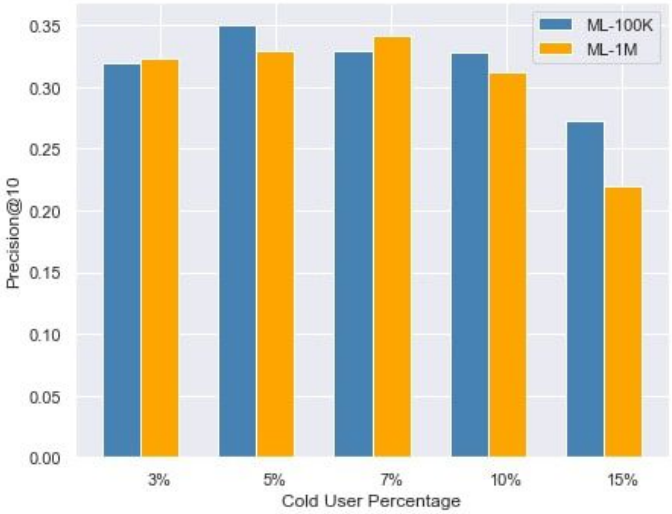


Figure 5.6: Cold User Threshold

achieved by the MDP-based approach is statistically significant over the majority of the baseline methods (with the exception of the P@10 measure on PopRank on ML-100K and SVD on ML-1M, respectively).

Table 5.4: Wilcoxon Signed P value against different baselines on Movielens 100k

MDP vs	P@10	R@10
Random	0.0008	0.0078
PopRank	0.0585	0.0100
LinUCB	0.0158	0.0089
CB	0.0093	0.0411
SVD	0.0002	0.0178
Deep MF	0.0067	0.0010
Neural CF	0.0312	0.0076

Table 5.5: Wilcoxon Signed P value against different baselines on Movielens 1M

MDP vs	P@10	R@10
Random	0.0014	0.0066
PopRank	0.0158	0.0074
LinUCB	0.0365	0.0289
CB	0.0027	0.0088
SVD	0.0721	0.0218
Deep MF	0.0011	0.0006
Neural CF	0.0043	0.0198

We deployed the Hybrid recommender system based on Reinforcement learning to the Movielens datasets in this paper. To overcome the cold start issue, we combined the benefits of two popular recommender system methodologies. The recommender system was modelled in the Reinforcement Learning framework. We integrated the Collaborative filtering technique into it by modelling the MDP states using Matrix factorization and utilized the neighborhood technique with the MDP framework for calculating the next item. We utilized a Content-based filtering approach for the cold user set, thereby improving the recommendation process further.

We are interested in expanding our work in recommender systems to other domains such as e-commerce products and deploying multi-agent reinforcement learning based recommender system frameworks to model the recommender system process in the future. Techniques that can lead to more diversity in the recommendation process could be the focus of future research. Using the correlation between the items, we can modify the action in our MDP architecture from the proposed approach to recommend more than one item at a time. Furthermore, with a multi-agent strategy, we can assess several scenarios of the recommendation process, each modelled by a different agent. We also believe that our method may be extended to a Deep Reinforcement Learning framework employing alternate architecture for Actor-Critic networks.