Reinforcement Learning

2nd Semester of 2021-2022

Ghadamiyan Lida
`lida.ghadamiyan@s.unibuc.ro`

Oanea Șmit Andrei
`smit.oanea@s.unibuc.ro`

Ouatu Bogdan Ioan
`bogdan.ouatu@s.unibuc.ro`

Popa Larisa
`larisa.popa2@s.unibuc.ro`

Chichirau Vlad-Vasile
`vlad.chichi.rau@s.unibuc.ro`

Ionescu Diana
`ionescu.diana2@s.unibuc.ro`

Abstract

The main purpose of this project was to tackle the problem of recommendation system in offline reinforcement learning. We followed two main approaches: interactive and sequential recommendation using two relevant data sets. We also highlight an under-explored problem, namely the offline evaluation of policies and present an empirical evaluation in the contextual bandit framework.

1 Introduction

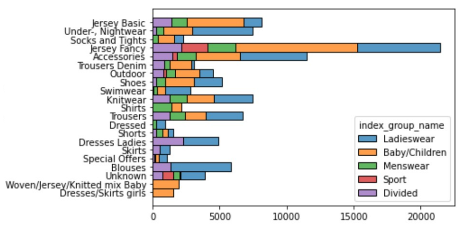
Recommender system is a popular task that implies presenting personalized information to users according to their history, behavior, interactions, preferences etc. Reinforcement learning focuses on learning by trial and error. In terms of scenarios, there are four types of recommendation systems in which reinforcement algorithms can be applied: interactive recommendation, conversational recommendation, sequential recommendation, and explainable recommendation. In an interactive recommendation scenario, as the name suggests, the agent constantly interacts with the user and receives a feedback. This feedback helps learning the policy that makes the agent send better recommendations. Conversational recommendation implies natural conversation between the agent and the user. The agent applies exploration process in which he recommends the items the user has not interacted with and finds out his opinion about them, the user's preferences. In a sequential recommendation system, based on the users historical interactions, the agent recommends the items that the user did not interact with and which may seem to be among the users preferences. The agent predicts users future preferences. Explainable Recommendation generates relevant explanations for recommendation results. Each type can be implemented using three types of reinforcement algorithms: value-function, policy search and actor-critic.

1.1 Used Datasets

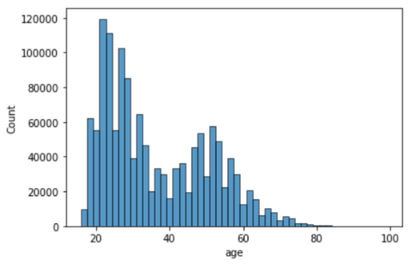
1.2 H & M Data Set

We used a public data set available on Kaggle in the "H&M Personalized Fashion Recommendations" contest. The data set consists in three csv files with information about customers, articles and transactions.

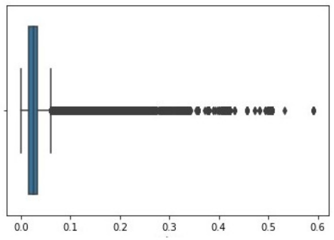
The articles file contains detailed metadata for each article that is available for purchase. The dataframe includes attributes as the description, the product name and various categories like garment group, product group, section name and index group. The plot below indicates some of the clothing types and how are the articles categorised. The garments grouped by index: The most frequent garment is jersey fancy which appears most in the women and children categories.



The customers dataframe contains the details of each user, more precisely, it contains a column with unique strings named *customer_id*, the columns *FN* and *active* contains value one or *missing*, some other features that a user has, are: the club member status, the *fashion_news_frequency* (how often a user reads a fashion article), the age and the postal code.
Based on the member status, we can observe from the figure that more than one million users are active and less than 500 of the total number of users have left the club. Regarding the age of the users, the vast majority of them have ages between 19 and 28, as we can observe from the following image.



The transactions data frame includes the customers and which articles did they bought, as well as the date and price. The next image illustrates a boxplot that describes the prices distribution. The box is defined by the first and third quantile and inside the box is the second. The quantiles are the values which divide the data set in four equal parts. Thus, the second one is the median. The whiskers represents the minimum an maximum values and the portion outside the box are the outliers.



1.3 Try at Home Data Set

This data set contains the history of purchases and returns generated by the "Try at Home" business model. "Try at Home" is a monthly subscription service that delivers garments to customers by mail. Customers do not choose what products they receive, instead they return the items that they do not purchase. The data is fairly sparse, with over 90 percent of active users having interacted with under 50 products of the over 1300 total products. The data set also contains item features such as category, color and fabric.

2 Base Line - Popularity Based Filtering

We implemented a simple baseline that predicts the products which are in trend at the moment based on the purchase history. Therefore, we recommended the first 12 most sold items to every user, achieving a score of 0.00304 in the competition leaderboard. The score did not change depending on the chosen period (one month, 3 moths or an year).

3 Approach

3.1 Sequential Recommendation

3.1.1 Actor-Critic

A similar problem with the one proposed by the previous data-set is treated in the following papers(from the same authors): *Self-Supervised Reinforcement Learning for Recommender Systems* [6] and *Supervised Advantage Actor-Critic for Recommender Systems* [7].

In the Related Work section, both papers talk about previously-existent approaches in the literature for Recommender Systems: RNNs, CNNs, and other Reinforcement Learning approaches. However, they propose two similar self-supervised reinforcement learning methods that use two types of neural network architectures that are going to be trained in paralel: the first type of network is trained to approximate the Q-values (Self-Supervised Qlearning, SQN and d Supervised Negative Q-learning, SNQN) and the other type uses the Q-values from the first (Self-Supervised Actor-Critic, SAC, and Supervised Advantage Actor-Critic, SA2C).

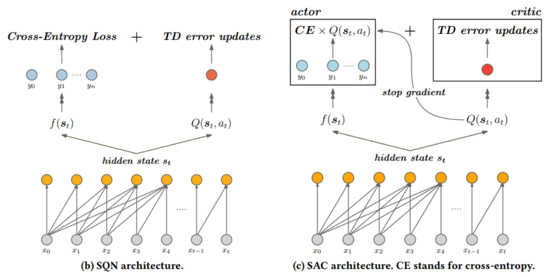
The initial architectures of the two types of proposes 2 frameworks, namely SQN and SAC that are described in the following figures.

Algorithm 1 Training procedure of SQN

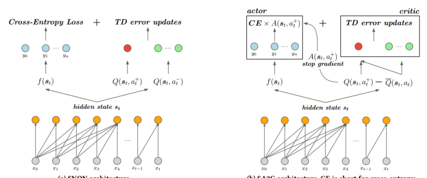
```
Input: user-item interaction sequence set X, recommendation model G, reinforcement head Q, supervised head
Output: all parameters in the learning space $\Theta$
Initialize all trainable parameters
Create G' and Q' as copies of G and Q, respectively
repeat
    Draw a mini-batch of $(x_{1:t}, a_t)$ from X, set rewards r
    $s_t$ = G($x_{1:t}$), $s_t$ = G'($x_{1:t}$)
    $s_{t+1}$ = G($x_{1:t+1}$), $s'_{t+1}$ = G'($x_{1:t+1}$)
```

Algorithm 2 Training procedure of SNQN

```
Input: user-item interaction sequence set X, recommendation model G($\cdot$), reinforcement head Q($\cdot$), supervised head f($\cdot$), pre-defined reward function r(s,a)
Output: all parameters in the learning space $\Theta$
Initialize all trainable parameters
Create G'($\cdot$) and Q'($\cdot$) as copies of G($\cdot$) and Q($\cdot$), respectively
repeat
    Draw a mini-batch of $(x_{1:t}, a_t^+)$ from X
    Draw negative actions set $N_t$ for $x_{1:t}$
    $s_t$ = G($x_{1:t}$), $s_t$ = G'($x_{1:t}$)
    $s_{t+1}$ = G($x_{1:t+1}$), $s'_{t+1}$ = G'($x_{1:t+1}$)
```



(b) SQN architecture. (c) SAC architecture. CE stands for cross-entropy.



(a) SNQN architecture. (b) SA2C architecture. CE is short for cross-entropy.

The main difference between those two papers is that the most recent one uses a negative sampling strategy for training. More precisely, the the RL head of SNQN is trained on both positive actions and a set of negative actions. This can be seen in their pseudo-codes for training the first type of networks (SQN and SNQN):

The loss formula used by both methods is:

$$L_s = -\sum_{i=1}^{n} Y_i log(p_i), \text{where } p_i = \frac{e^{y_i}}{\sum_{i'=1}^{n} e^{y_{i'}}}.$$

Where Yi is 1 if the user interacted with the i-th item in the next timestamp, and 0 otherwise.

After they integrate the two types of networks with 4 state-of-the-art sequential recommendation architectures (GRU, Caser, NItNet and SASRec), they show in the results section that the new architecture, which uses negative sampling, achieves better results on both of the two metrics used: Hit Ratio when it comes to clicks done by the user on a specific item (HR) and Normalized Discounted Cumulative Gain (NDCG).

3.1.2 Actor-Critic - Experiment

We tried a simple experiment using the Actor-Critic method, where both the Actor and the Critic are implemented as neural networks.

First we created a simple Environment. The Environment get as input all the files available in the data-set and creates internal data frames that stores all existing customer, items and transaction history. More so, for the customer and the items converted into a 30-dimensional embedding space.

The internal state of the Environment is defined by a *current position*. This *current position* is used to mark the position of the current instance of the Environment in the transaction history.

The **step** function in the Environment has as arguments the action, where the action is defined as a list of 12 items that represent the current recommendation. For this action we retrieve the current 'purchased' item from the transaction history at the *current position*. Next, the **reward** is computed, where the reward is defined as the sum of the Cosine Similarity of the actual 'purchased' item with all the recommended items(action).

For the item embedding we used the truncated sentence embedding of the item description as it gave better result that the initial random embedding of the items. For the customer we chose a random embedding in the previous defined embedding dimension.

We used a simple Actor architecture, where it gets as input the user embedding, and outputs the action(the 12 items).

We used a simple Critic architecture, where it gets as input the user embedding and 12 items, and outputs the Q-value.

We trained the model for a small fraction of the original transaction history because of time constrain(the entire history would have taken 90 days without optimisations). We achieved a score of 0.00064 in the competition leader-board.

3.2 Policy Search -KERL

[5] was presented during an event in China that took place in July 25–30, 2020. It proposes a Knowledge-guided Reinforcement Learning (KERL) framework. The paper's purpose is to create a reinforcement learning model that makes use of knowledge graphs. The task is to predict the next item the user will interact with given the user history and knowledge graph.

The modeling process is designed as a Markov Decision Process (MDP). At each time step, the process is in some state. The environment's state contains information regarding the users history interactions and information from KG. The KG contains records of triplets: information about the users and items, the relation between them(the interaction). The information over the items is encoded as an embedding. For item embeddings generation it was used TransE method from OpenKE library. The information from each state also can be encoded as an embedding. In order to include knowledge information in the state representations there were used two kinds of state representations: sequence-level and knowledge-level state representations. Sequence-level state representation implies a standard recurrent neural network for encoding the history sequence of interactions. The purpose of this representation is to create sequential characteristics of user preference. This step does not imply using knowledge information. In the knowledge-level state representation there is considered both exploitation and exploration by modeling two kinds of knowledge-based preference: current knowledge-based preference and future knowledge-based preference. The current knowledge-based preference is represented by the average pooling of the items embeddings the user has interacted with. For the exploration part in predicting the future items, a neural network using a Multi-Layer Perception was built whose input is the current preference representation. At each time step, a new item preference representation is predicted. In other words, knowing the current state, a new item is predicted. For the final state representation, sequence-level information, the existing knowledge characteristics for achieving knowledge-based exploitation, the future preference prediction for achieving knowledge-based exploration are all combined. This whole process leads to a good trade-off between exploitation and exploration.

In every state, the agent performs an action modeled by a policy. This policy takes as input the state and outputs a distribution over all possible actions. The action is to select an item at time t+1 for recommendation. The probability of selecting an item is computed as Softmax over item embeddings and state embeddings and some parameter. After each action, the agent obtains an intermediate reward, which is defined by integrating sequence-level and knowledge-level rewards and the new state is updated. In sequential recommendation, the overall reward is usually measured based on the exact match for item IDs. The sequence level reward's purpose is to generate consistent m-grams from the actual sequence. The reward in this part is computed using bleu score. In the knowledge level the focus, we measure using cosine similarity how close the predicted subsequence embeddings is to the actual subsequence embeddings. The final reward considers both the sequence-level and knowledge-level match. It is also used gamma discount factor to balance the importance of the current reward and future reward.

The training purpose is to learn a stochastic policy that maximizes the expected cumulative reward for all uses. The KERL training consists :

- Input: Policy, reward, knowledge graph, learning rate.

- The parameters to learn are randomly initialized.
- The items embeddings are generated using TransE.
- Take all users and at each step (the number of time steps is predefined) obtain the history sequence(recurrent neural network of historical interactions), obtain the current state (average of items embeddings) and compute the future preference (neural network having as input the current state), all 3 being combined (concatenated)I n a vector state.
- For a predefined number of sub sequences, sample a subsequence, compute the policy according to the vector state and update the learnable parameters. Then, apply the subsequence to the induction network in order to predict the future preference according to the current state, and only then update the vector state using the same formula(history concatenated with current state concatenated future state). The induction network consists as mentioned before in a neural network, namely it is a linear regression model using MSE loss.
- The process is repeated as long as better values are estimated.

There were used 4 datasets. The users and items with less than some number of interactions were deleted. The records are sorted. According to all evaluation metrics, the knowledge based methods perform better than sequential-based methods.

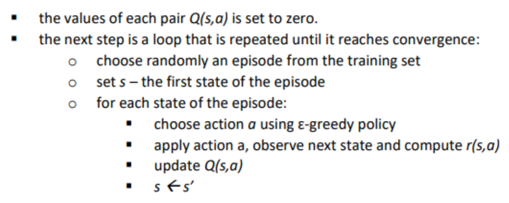
3.3 Value function - UWR

The method introduced in this paper [4] is in the interactive recommendation scenario which means that the user gives, repeatedly, feedback from which the agent will learn and update the policy function. [2]
The problem of recommendation is modeled as a Q-Learning problem and it uses as information the logs of the web usage. After designing the state, action and the rewarding policy, the model is trained on the logs training set, at this step, the system learns to recommend items.
Problem. Just like in a clothes recommending system, the problem is to predict a few pages that the user of the website will watch next, knowing the pages that this user requested until this step. A page is considered to be successful if, until the last step of this session, the user chose to watch it (for example, if the web page is recommended at the first step and it is chosen at the i-th step –considering that a session contains more than i steps, that page is considered to be a good recommendation).
Q-Learning. The value function Q(s,a), in Q-Learning, represents the value of making the action a in the given state s. This value is computed as the discounted sum of the future rewards that may get by making the action a. Solving this problem with Q-Learning implies choosing the suited definitions for states and actions, and also implies finding a correct reward function and to develop a training procedure for the system.
Some observations were made regarding the problem: from the current state and action, it cannot compute deterministically the next state since the user may choose from a large number of pages, fact that influence the learning process. And the actions of the system coincide with the predictions. Regarding the reward function, an action will be rewarded if it is visited at the current step or later.
The states will be computed as N-grams (due to the possibility of having to deal with infinite number of states), meaning that only the last w pages that the user requested are taken into consideration and, following the same pattern, the recommended pages' sequence take into consideration the last w' recommendations. One *action* is defined, at one step, to be one web page recommendation. The actions should be rewarded in each state, reported to the consequent state (named after-state) which resulted from the agent action and from factors that the environment cannot control.
The *reward* is depended on the after-state, actually it is depended on the intersection between all the recommendations items that were made in each state until current state, and the current page sequence of the state [4]. In order to reward one correct recommendation at a step (and not to give a reward for all the correct recommendations given until now) it will be rewarded the action preformed at the state s_i considering the last page visited in the recommended pages list in state s_{i+1}. Also an item that was recommended earlier than when the user choose the same page, it will get a greater reward (the goal of the problem is for the user to spend less time in searching the wanted web page and recommending it earlier is a good result). Taking into consideration the problem, if the user spends a lot of time on a specific page it means that is a good page, otherwise, it is a less interesting page – this aspect makes the time that the user spends reading an important factor for the rewarding function and it will be taken into consideration.

Training. The Q-Learning algorithm tries to estimate the Q-values – the evaluation of preforming an action in a state – however, it is not important for the problem to get the accumulative rewards that can be achieved from each state. Actually, each action is a recommendation and the value of each action represents an estimation of how good is the recommendation. The rule for updating the Q-values that was chosen considers the fact that the action that is made in a state can give different results, is:

$$Q_{(s,a} = (1-\alpha_n)Q_{n-1}(s,a) + \alpha_n[r(s,a) + \\ + \gamma \max_{a'} Q_{n-1}(\delta(s,a),a')] \quad (1)$$

The dataset is split in episodes, without any recommendations and the training works as follows:



- the values of each pair *Q(s,a)* is set to zero.
- the next step is a loop that is repeated until it reaches convergence:
    - choose randomly an episode from the training set
    - set *s* – the first state of the episode
    - for each state of the episode:
        - choose action *a* using ε-greedy policy
        - apply action a, observe next state and compute *r(s,a)*
        - update *Q(s,a)*
        - *s* ← *s'*

For this problem, the system converges after 3000-5000 repeating each episode and one change was made to the reward function: a recommendation will be rewarded if it was already visited.

3.4 Interactive Recommendation

3.4.1 Multi-armed bandits

Multi-armed bandits are [1] analyzes how to evaluate a policy in an offline fashion, given that you know the policy which was used to generate the data.

The bandit setting is a specific form of reinforcement learning where a learner chooses action $a \in A$ and the world reacts with reward $r_a \in [0, 1]$

In the contextual bandit setting, on which the paper is based on, the world generates a context $x \in X$, of which the learner is aware, prior to the learner choosing an action.

The rewards are known only for the actions the policy has taken. In this setting, the urge to apply supervised learning to generate a new policy is misguided since you could only optimize on the data produced by the old policy, thus being stuck forever in a local minima generated by the old policy (Experiment 3.1). To solve this, a way of estimating the total cost of the new policy is needed. There are three main approaches: - Direct Method - approximate the reward for unseen (x, a) pairs, where x is a context and a is the action - Inverse Propensity Score - approximate the probability $\mu(x, a)$ of the old policy selecting action a in the context x; this approach generates an unbiased estimator - Doubly robust estimator - combines the previous two approaches to generate an unbiased estimator that should have a lower variance if the reward estimation is good enough

Using one of the estimators, we can than apply the technique detailed in [3]. First, we model the old policy as probabilistic. Then we estimate how likely the old policy is to chose some reward. For each event, we create a synthetic contextual bandit event based on the context, action taken, reward and the estimated probability of chosing some action. Now we can apply an offline contextual bandit algorithm.

While the main benefit of this approach is the possibility to evaluate policies offline, it's very hard to compare one evaluation strategy with another without deploying the policies and seeing how well the offline evaluation went. Experiment 3.4.2 details the offline evaluation strategy with the offline bandit training on the Try at Home data set and evaluates the results on the biased test set. While the results should not be taken for granted, based on domain knowledge, they should be a relatively decent approximation of the models' performance.

3.4.2 Collaborative filtering Experiment

A matrix factorization-based collaborative filtering model was trained on Try at Home data set on 70 percent of the data. Only users with over 20 item interactions were considered. It was than tested on the next 30 percent. Once tested, the users were simulated to either buy or return items at random and than the newly generated data was attached to the training data. The model was than retrained on both the real and synthetic data. This step was repeated for 10 iterations.

Results:

It generally takes only one iteration for the number of distinct products to drop from 60 +- 5 products to 30+-3 and than the number of distinct items remains under 30. The same popular items are sent every iteration.

3.4.3 Bandit Experiment

The bandit design is as follows: items constitute contexts, possible actions is to send an item to one of the users and the reward is 1 if the user keeps the product, or 0 otherwise. We estimated the old policy by setting a fixed probability to every kept item. We sample 500 users with over 20 interactions since the process is pretty slow. We use the Vowpal Wabbit library, provided by the author of the paper [3]. We then proceed training using a Doubly Robust bandit, using the item features present in the dataset. Now, we predict which item should go to which customer. Now, for every user we group together the items that should get to them. We then select the most popular 12 bandit recommendations or pad users that don't have enough recommendations with the most popular items. We than compute the MAP@12 score.

We also analyze product popularity:

Analysis of the results: Although this method fails to generate good results on the classical evaluation procedure, it is to be expected since the sparsity of the data makes the bandits hyper-exploratory. Also, we are optimizing for sending the most number of items to users, but evaluate on number of kept items. Since the test data is sparse, the chance of the new policy sending an item already sent by the old policy is low, if we do not send popular items. Until the strategy is deployed, this result may be the best proxy for the evaluation performance of this strategy.

Method	Result
Send only the most popular	0.016
Collaborative filtering	0.015
Bandit approach	0.011

No. items in top 12 recommended/user	Result
Send only the most popular	1
Collaborative filtering	0.78
Bandit approach	0.62

4 Conclusions and Future Work

We documented and analyzed broadly the literature on reinforcement learning methods to address the recommendations problem. We generated two main empirical analysis approaches. We implemented a RL algorithm able to predict user preferences on the HM data set. We highlighted the offline evaluation bias problem that is seldom approached in literature, even though it has very important practical applications. We modeled the Try at Home problem as a bandit problem and applied the methods presented in [3].

Future work may be focused on pursuing offline policy evaluation methods in a broader RL framework than bandits. This may result in algorithms with superior predicting power since classical supervised learning has a hard time correcting for selection bias.

Bibliography

References

[1] SANG HOON KIM. Offline policy evaluation in a contextual bandit problem.

[2] Yuanguo Lin, Yong Liu, Fan Lin, Pengcheng Wu, Wenhua Zeng, and Chunyan Miao. A survey on reinforcement learning for recommender systems. *arXiv preprint arXiv:2109.10665*, 2021.

[3] Alex Strehl, John Langford, and Sham M Kakade. Learning from logged implicit exploration data. 23, 2010. https://arxiv.org/pdf/1003.0120.pdf.

[4] Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 113–120, 2007.

[5] Pengfei Wang, Yu Fan, Long Xia, Wayne Xin Zhao, Shaozhang Niu, and Jimmy Huang. *KERL: A Knowledge-Guided Reinforcement Learning Model for Sequential Recommendation*. Association for Computing Machinery, New York, NY, USA, 2020.

[6] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. Self-supervised reinforcement learning for recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 931–940, 2020.

[7] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. Supervised advantage actor-critic for recommender systems. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 1186–1196, 2022.