APPLIED MARKOV DECISION PROCESS AND REINFORCEMENT LEARNING

MINI REPORT - I

on

A Deep Reinforcement Learning Based Long-Term Recommender System

Submitted By :
Ashirbad Sarangi
SC23M002

Under Esteemed Guidance of :
Dr. Vineeth B. S.
ASSISTANT PROFESSOR

Indian Institute of Space Science and Technology (IIST),
India
March 15, 2024

1 Problem Statement and Solution

The main aim of the recommender systems are to maximise the overall accuracy of the long-term recommendations for the user. However, according to [1] there are certain limitations that exist for most of the recommenders that exist presently such as :

1.1 Limitations of previous recommenders

1. Most of the existing recommenders use the collaborative filtering based matrix factorisation method which adopts a static view during recommendation process and ignores the dynamic and sequential nature of the recommendation problem. The recommender doesnt evolve as the user choice preferences change and keeps on repeating the recommendations.
2. As the recommender depends entirely upon the historical data of the user, it suffers from the cold start problem.
3. Mostly recommenders fail to exploit the sequential decision nature of the recommendations. And out of the RNN based recommenders present, most of them aim for short term recommendation which maximises the immediate reward and does not guarantee optimal long-term reward.

1.2 Advantages of the proposed recommender

1. The ability of long term recommender. Using the Markov Decision Process (MDP) in the process helps to select the recommendation list in order to get the optimal reward.
2. User state can be dynamically updated. As reinforcement learning is used, the state of the user depends upon the action taken by him/her in previous time slot. The recommender reacts to the feedback it receives and generates the list accordingly for the next recommendation.
3. Suitable for both cold start and warm start model without additional content information. The use of RNN helps in generate the probability transition matrix which learns exclusively for the user/

2 Assumptions

1. Each user u has a total of $|I_u|$ rounds of interactions with his/her personal recommendation agent.
2. u can select at most one item in $P^N_{u,t}$ at each time t.
3. At each time t, if $P^N_{u,t} \cap I_u = \phi$, user u should select only one item $\hat{a}_{u,t} \in P^N_{u,t} \cap I_u$ with the highest estimated probability, and responds a positive feedback
4. At each time t, if $P^N_{u,t} \cap I_u = \phi$, there is no hit item, and user u responds a negative feedback.
5. If i $\in P^N_{u,t} \cap I_u$ is selected, it should be removed from I_u to avoid being repeatedly chosen.

where

Notation	Description
I	the set of all items
I_u	the subset of I, which includes items that user u likes
$s_{u,t}$	the t^{th} state of RNN for user u
$P^N_{u,t}$	the t^{th} recommendation list including N items for user u
$f_{u,t}$	the t^{th} feedback responded by user u
$V_{u,t}$	the t^{th} immediate reward with respect to $f_{u,t}$
$R_{u,t}$	the t^{th} accumulative reward with respect to $f_{u,t}$

3 Procedure

3.1 Data Collection and Preprocessing

The MovieLens100K data set was used to make this Markov Decision Process. This dataset contains the information of :

- 100,000 ratings (1-5) from 943 users on 1682 movies.
- Movies are classified into 19 genres
- Each user has rated at least 20 movies.
- Simple demographic info for the users (age, gender, occupation, zip)

The data was initially present segregated into mainly three different files - *u.user, u.item, u.data*. Owing to the large scale of the data it is scaled down to simpler dimensions by the following process :

1. The ratings of the was centralised.
2. From the *u.item* dataset, all the non-essential headers were dropped and only the necessary details of the movie and genre information is retained.
3. The user with maximum positive ratings was picked. Accordingly, only the movies rated by her were filtered out. A combined analytical dataset is created containing userid, movieid, movietitle and genres are kept and saved in another csv file which is used as the ultimate dataset for the MDP.

The recommender system is designed to provide personal recommendation to a particular user only. This is equivalent to select only one user of the many users present. Basically, of the 943 users, the focus is now shifted only one particular customer, who is 35 years old female working as an educator who had rated 540 movies.

3.2 Data Preparation

This step takes the csv file created in 3.1. Currently there are 18 genres available for each movie selected by the user. Due to the high complexity of computation, the data was further reduced to the two most rated genres by the user : *Drama* , *Comedy*. Thus this is I. There was 4 classes made based on the genres :

Class	Drama	Comedy
0	0	0
1	0	1
2	1	0
3	1	1

3.3 Declarations and Definitions

The interested set of user I_u was found from the I.

$$f(i) = \begin{cases} 1 & \tilde{\mathrm{R}}(\mathrm{i}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

After the interested set is found out, then the feedback function is defined as :

$$f_{u,t} = \mathbb{F}(P_{u,t}^N, I_u) = \begin{cases} 1 & P_{u,t}^N \cap I_u \neq \phi \\ 0 & \text{otherwise} \end{cases}$$

The state space s_t can be defined as the number of movies at the time instant t in the recommended list $P_{u,t}^N$. As $N = 3$, and there are 4 classes. It can be equivalently imagined as solving a linear combination of 4 non-negative integers which add up to 3. Thus there are $\binom{6}{3} = 20$ states. Thus the state space is discrete in nature. The *transition* function can be defined as :

1. New list is created with 60% weightage to the liked movies in the recommended list and 40% weightage to old list.
2. Once the new list is available it is normalised and discretised to know which class has how many occurences or in other which words which is the next state/
3. In case the recommended was not at all liked by the user then the system switches back to the previous state.

The action is also discretised as this will have N movies at any time t (i.e. $P_{u,t}^N$). The *generate* function can be defined as a complex series of functions mathematically. But basically, it takes input as current state and the previously recommended movies for masking them so that they dont appear in the immediate next recommendation. The function can be explained in the following steps :

1. Based on the current state, randomly movies are picked as candidates from each class corresponding to their occurences in the state, given that the movies are available for selection i.e. are unmasked.
2. In case , the number of movies selected are not equal to N, then randomly unmasked movies are selected to fill in the shortage.
3. Now the exploration and exploitation in such a way that 8% times exploration is done by randomly selecting N movies from the candidate list, else rest of the times, the top N movies are selected.
4. Finally, masking is done of the final selection of candidate list and the movie ids are stored for use in the next time t

The reward function can be defined as :

$$V_{u,t} = \begin{cases} 1 & f_{u,t} > 0, \\ -0.2 & \text{otherwise} \end{cases}$$

$$R_{u,t} = \sum_{k=0}^{M-k} \gamma^k V_{u,t+k}$$

where $\gamma \in (0,1)$

3.4 Policy Iteration and Value Iteration

Algorithm 1 Policy Iteration

$restart()$
while $t \le |I_u|$ **do**
  $t \leftarrow t + 1$
  $f_{u,t} \leftarrow \mathbb{F}(P^N_{u,t}, I_u)$
  $state \leftarrow transition(P^N_{u,t}, oldstate)$
  $P^N_{u,t} \leftarrow generate(state, oldmovies)$
  $R_{u,t} = \sum_{k=0}^{M-k} \gamma^k V_{u,t+k}$
end while

Algorithm 2 Value Iteration

$restart()$
while $t \le |I_u|$ **do**
  $t \leftarrow t + 1$
  $f_{u,t} \leftarrow \mathbb{F}(P^N_{u,t}, I_u)$
  for $tempstate$ in $allstates$ **do**
    find reward for tempstate
  end for
  $P^N_{u,t} \leftarrow generate(state, oldmovies)$
  $R_{u,t} = \sum_{k=0}^{M-k} \gamma^k V_{u,t+k}$
end while

4 Exploratory Data Analysis

[1]:
```
import pandas as analytics
import numpy as maths
import os
```

[2]:
```
source_path = 'MovieLens100k_dataset'
data_path = os.path.join(source_path,'u.data')
genre_path = os.path.join(source_path,'u.genre')
```

```
item_path = os.path.join(source_path,'u.item')
occupation_path = os.path.join(source_path,'u.occupation')
user_path = os.path.join(source_path,'u.user')
```

[3]:
```
def extract_values(a):
    return [i.strip().replace(" ","_") for i in a.split("|")]
```

[4]:
```
data_headers = """user id | movie id | rating | timestamp"""
item_headers = """movie id | movie title | release date | video release date |
              IMDb URL | unknown | Action | Adventure | Animation |
              Children's | Comedy | Crime | Documentary | Drama | Fantasy |
              Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi |
              Thriller | War | Western |"""
user_headers = """user id | age | gender | occupation | zip code"""


data_headers = extract_values(data_headers)
item_headers = extract_values(item_headers)[:-1]
user_headers = extract_values(user_headers)
genres = analytics.read_csv(genre_path,sep="|")['unknown'].tolist()
```

[5]:
```
df_data = analytics.read_csv(data_path,header = None,sep="\t",names =␣
 ↪data_headers)
df_data['rating'] = df_data['rating'] - 3
df_data
```

[5]:

	user_id	movie_id	rating	timestamp
0	196	242	0	881250949
1	186	302	0	891717742
2	22	377	-2	878887116
3	244	51	-1	880606923
4	166	346	-2	886397596
...	...	...	...	...
99995	880	476	0	880175444
99996	716	204	2	879795543
99997	276	1090	-2	874795795
99998	13	225	-1	882399156
99999	12	203	0	879959583

[100000 rows x 4 columns]

[6]:
```
df_users = analytics.read_csv(user_path,header=None,sep = "|",names =␣
 ↪user_headers)
df_users
```

[6]:

	user_id	age	gender	occupation	zip_code
0	1	24	M	technician	85711

```
1          2   53      F          other    94043
2          3   23      M         writer    32067
3          4   24      M     technician    43537
4          5   33      F          other    15213
..       ...  ...    ...            ...      ...
938      939   26      F        student    33319
939      940   32      M  administrator    02215
940      941   20      M        student    97229
941      942   48      F      librarian    78209
942      943   22      M        student    77841

[943 rows x 5 columns]
```

[7]:
```python
df_items = analytics.read_csv(item_path, header = None, sep = "|" , names =␣
 ↪item_headers)
df_items = df_items.drop(['release_date','video_release_date','IMDb_URL'],axis =␣
 ↪1)
df_items = df_items[df_items['unknown'] == 0]
df_items = df_items.drop('unknown',axis = 1)
df_items
```

[7]:
```
      movie_id                                movie_title  Action  Adventure  \
0            1                           Toy Story (1995)       0          0
1            2                           GoldenEye (1995)       1          1
2            3                          Four Rooms (1995)       0          0
3            4                          Get Shorty (1995)       1          0
4            5                             Copycat (1995)       0          0
...        ...                                        ...     ...        ...
1676      1678                           Mat' i syn (1997)       0          0
1677      1679                            B. Monkey (1998)       0          0
1678      1680                        Sliding Doors (1998)       0          0
1679      1681                         You So Crazy (1994)       0          0
1680      1682  Scream of Stone (Schrei aus Stein) (1991)       0          0

      Animation  Children's  Comedy  Crime  Documentary  Drama  Fantasy  \
0             1           1       1      0            0      0        0
1             0           0       0      0            0      0        0
2             0           0       0      0            0      0        0
3             0           0       1      0            0      1        0
4             0           0       0      1            0      1        0
...         ...         ...     ...    ...          ...    ...      ...
1676          0           0       0      0            0      1        0
1677          0           0       0      0            0      0        0
1678          0           0       0      0            0      1        0
1679          0           0       1      0            0      0        0
1680          0           0       0      0            0      1        0
```

```
      Film-Noir  Horror  Musical  Mystery  Romance  Sci-Fi  Thriller  War  \
0             0       0        0        0        0       0         0    0
1             0       0        0        0        0       0         1    0
2             0       0        0        0        0       0         1    0
3             0       0        0        0        0       0         0    0
4             0       0        0        0        0       0         1    0
...         ...     ...      ...      ...      ...     ...       ...  ...
1676          0       0        0        0        0       0         0    0
1677          0       0        0        0        1       0         1    0
1678          0       0        0        0        1       0         0    0
1679          0       0        0        0        0       0         0    0
1680          0       0        0        0        0       0         0    0

      Western
0           0
1           0
2           0
3           0
4           0
...       ...
1676        0
1677        0
1678        0
1679        0
1680        0

[1679 rows x 20 columns]
```

[8]:
```
df_users
```

[8]:
```
     user_id  age gender     occupation zip_code
0          1   24      M     technician    85711
1          2   53      F          other    94043
2          3   23      M         writer    32067
3          4   24      M     technician    43537
4          5   33      F          other    15213
..       ...  ...    ...            ...      ...
938      939   26      F        student    33319
939      940   32      M  administrator    02215
940      941   20      M        student    97229
941      942   48      F      librarian    78209
942      943   22      M        student    77841

[943 rows x 5 columns]
```

[9]:
```
df_data
```

[9]:
```
       user_id  movie_id  rating  timestamp
0          196       242       0  881250949
1          186       302       0  891717742
2           22       377      -2  878887116
3          244        51      -1  880606923
4          166       346      -2  886397596
...        ...       ...     ...        ...
99995      880       476       0  880175444
99996      716       204       2  879795543
99997      276      1090      -2  874795795
99998       13       225      -1  882399156
99999       12       203       0  879959583

[100000 rows x 4 columns]
```

[10]:
```python
df_data.sort_values('user_id')
```

[10]:
```
       user_id  movie_id  rating  timestamp
41842        1        46       1  876893230
38751        1       257       1  874965954
8976         1        12       2  878542960
3248         1        74      -2  889751736
3260         1       134       1  875073067
...        ...       ...     ...        ...
95594      943       217       0  888640067
77956      943        94       1  888639929
76855      943       943       2  888639614
94966      943       566       1  888639886
90134      943         2       2  888639953

[100000 rows x 4 columns]
```

[11]:
```python
# max_user_id = df_data['user_id'].value_counts(ascending = False).reset_index().
↪set_index('count').sort_values(by= 'count',ascending = False).
↪iloc[0]['user_id'].tolist()
max_user_id = df_data.groupby('user_id').agg({'rating':lambda x:x.sum()}).
↪reset_index().sort_values(by = 'rating',ascending = False)['user_id'].iloc[0]
df_users = df_users[df_users['user_id'] == max_user_id]
df_users
```

[11]:
```
     user_id  age gender occupation zip_code
449      450   35      F   educator    11758
```

[12]:
```python
df_ratings = df_data[df_data['user_id'] ==␣
↪max_user_id][['user_id','movie_id','rating','timestamp']]
df_ratings
```

[12]:
```
       user_id  movie_id  rating  timestamp
17656      450       470       2  887139517
17680      450       783       0  882399818
17764      450      1147       1  882374497
17963      450       100       1  882374059
18055      450        58       0  882373464
...        ...       ...     ...        ...
98566      450       584       2  882397223
98871      450       732       0  882395662
99039      450       388       0  882471604
99614      450      1490       0  882396929
99772      450       654       1  882373928

[540 rows x 4 columns]
```

[13]:
```
df_items = df_items[df_items['movie_id'].isin(df_data['movie_id'])]
df_items
```

[13]:
```
      movie_id                                movie_title  Action  Adventure  \
0            1                           Toy Story (1995)       0          0
1            2                           GoldenEye (1995)       1          1
2            3                          Four Rooms (1995)       0          0
3            4                          Get Shorty (1995)       1          0
4            5                             Copycat (1995)       0          0
...        ...                                        ...     ...        ...
1676      1678                          Mat' i syn (1997)       0          0
1677      1679                           B. Monkey (1998)       0          0
1678      1680                       Sliding Doors (1998)       0          0
1679      1681                        You So Crazy (1994)       0          0
1680      1682  Scream of Stone (Schrei aus Stein) (1991)       0          0

      Animation  Children's  Comedy  Crime  Documentary  Drama  Fantasy  \
0             1           1       1      0            0      0        0
1             0           0       0      0            0      0        0
2             0           0       0      0            0      0        0
3             0           0       1      0            0      1        0
4             0           0       0      1            0      1        0
...         ...         ...     ...    ...          ...    ...      ...
1676          0           0       0      0            0      1        0
1677          0           0       0      0            0      0        0
1678          0           0       0      0            0      1        0
1679          0           0       1      0            0      0        0
1680          0           0       0      0            0      1        0

      Film-Noir  Horror  Musical  Mystery  Romance  Sci-Fi  Thriller  War  \
0             0       0        0        0        0       0         0    0
1             0       0        0        0        0       0         1    0
```

```
2              0       0        0        0        0       0          1    0
3              0       0        0        0        0       0          0    0
4              0       0        0        0        0       0          1    0
...          ...     ...      ...      ...      ...     ...        ...  ...
1676           0       0        0        0        0       0          0    0
1677           0       0        0        0        1       0          1    0
1678           0       0        0        0        1       0          0    0
1679           0       0        0        0        0       0          0    0
1680           0       0        0        0        0       0          0    0

      Western
0           0
1           0
2           0
3           0
4           0
...       ...
1676        0
1677        0
1678        0
1679        0
1680        0

[1679 rows x 20 columns]
```

[14]:
```
df_rated_items = df_items.merge(df_ratings,on='movie_id',how = 'inner')
req_order = ['user_id','movie_id','movie_title','rating'] + genres +␣
 ↪['timestamp']
df_rated_items = df_rated_items[req_order]
```

[15]:
```
df_rated_items.to_csv('rated_movies.csv',index= False)
```

5 Recommender

[1]:
```
import pandas as analytics
import numpy as maths
import warnings
import time
warnings.filterwarnings("ignore")
```

[2]:
```
df_rated_movies = analytics.read_csv('rated_movies.csv')
user_id = df_rated_movies['user_id'].unique()[0]
df_rated_movies = df_rated_movies.drop(['user_id','timestamp'],axis = 1)
df_rated_movies
```

[2]:
```
     movie_id               movie_title  rating  Action  Adventure  \
0           1          Toy Story (1995)       1       0          0
1           2          GoldenEye (1995)       1       1          1
2           3         Four Rooms (1995)       1       0          0
3           4         Get Shorty (1995)       0       1          0
4           7     Twelve Monkeys (1995)       1       0          0
..        ...                       ...     ...     ...        ...
535      1480  Herbie Rides Again (1974)      0       0          1
536      1490             Fausto (1993)       0       0          0
537      1518      Losing Isaiah (1995)       1       0          0
538      1521      Mr. Wonderful (1993)       0       0          0
539      1603             Angela (1995)       0       0          0

     Animation  Children's  Comedy  Crime  Documentary  ...  Fantasy  \
0            1           1       1      0            0  ...        0
1            0           0       0      0            0  ...        0
2            0           0       0      0            0  ...        0
3            0           0       1      0            0  ...        0
4            0           0       0      0            0  ...        0
..         ...         ...     ...    ...          ...  ...      ...
535          0           1       1      0            0  ...        0
536          0           0       1      0            0  ...        0
537          0           0       0      0            0  ...        0
538          0           0       1      0            0  ...        0
539          0           0       0      0            0  ...        0

     Film-Noir  Horror  Musical  Mystery  Romance  Sci-Fi  Thriller  War  \
0            0       0        0        0        0       0         0    0
1            0       0        0        0        0       0         1    0
2            0       0        0        0        0       0         1    0
3            0       0        0        0        0       0         0    0
4            0       0        0        0        0       1         0    0
..         ...     ...      ...      ...      ...     ...       ...  ...
535          0       0        0        0        0       0         0    0
536          0       0        0        0        0       0         0    0
537          0       0        0        0        0       0         0    0
538          0       0        0        0        1       0         0    0
539          0       0        0        0        0       0         0    0

     Western
0          0
1          0
2          0
3          0
4          0
..       ...
535        0
```

```
536         0
537         0
538         0
539         0

[540 rows x 21 columns]
```

[3]:
```
genres = list(df_rated_movies.drop(['movie_id','movie_title','rating'],axis = 1).
 ↪columns)
selected_genres = df_rated_movies[genres].sum().sort_values(ascending = False).
 ↪reset_index(drop = False).iloc[:2]['index'].tolist()
print("Most Popular user #",user_id," are ",selected_genres,"\n")

df_rated_movies['class'] = df_rated_movies[selected_genres[0]] * 2 +␣
 ↪df_rated_movies[selected_genres[1]]
class_mapping = df_rated_movies[selected_genres + ['class']].drop_duplicates().
 ↪sort_values('class').set_index('class')     # mapping of drama ,comedy movie to␣
 ↪respective class
classes = class_mapping.index.tolist()

selected_columns = ['movie_id','rating','class']
df_rated_movies = df_rated_movies[selected_columns]
df_rated_movies
```

```
Most Popular user # 450  are  ['Drama', 'Comedy']
```

[3]:
```
     movie_id  rating  class
0           1       1      1
1           2       1      0
2           3       1      0
3           4       0      3
4           7       1      2
..        ...     ...    ...
535      1480       0      1
536      1490       0      1
537      1518       1      2
538      1521       0      1
539      1603       0      2

[540 rows x 3 columns]
```

[4]:
```
N = 3
```

[5]:
```
df_interested = df_rated_movies.copy()
df_interested['F'] = df_interested['rating'].apply(lambda x : 1 if x > 0 else 0)
df_interested = df_interested[df_interested['F'] > 0].drop('F',axis=1)
```

```
df_interested
```

[5]:

```
     movie_id  rating  class
0           1       1      1
1           2       1      0
2           3       1      0
4           7       1      2
5          10       1      2
..        ...     ...    ...
529      1425       1      3
530      1435       1      1
532      1444       1      1
533      1446       1      1
537      1518       1      2

[378 rows x 3 columns]
```

[6]:

```
def feedback(P,feedback_values, t):                     # instead of actual fut␣
↪definition, its changed a bit, i.e. if in the selected list there is a movie␣
↪user likes then it is feedback is 1 else 0
    value = 0
    if (P['rating'] > 0).any() :
        value = 1
    feedback_values.insert(t,value)
    return feedback_values

def generate(state,old_movies):
    movie_ids = []
    for _class in state.index :
        df_temp = df_rated_movies[(df_rated_movies['mask'] ==␣
↪1)][df_rated_movies['class'] == _class].sample(n = int(state[_class])+1)
        movie_ids = movie_ids + df_temp['movie_id'].to_list()

    if len(movie_ids) < N :
        diff = N - len(movie_ids)
        additional_ids = df_rated_movies[df_rated_movies['mask'] ==␣
↪1][~df_rated_movies['movie_id'].isin(movie_ids)].sample(n = diff)['movie_id'].
↪tolist()
        movie_ids = movie_ids + additional_ids
    df_candidates = df_rated_movies[df_rated_movies['movie_id'].isin(movie_ids)]

    if maths.random.random() > 0.08 :                        # exploitation
        df_candidates = df_candidates.iloc[:N]
    else : df_candidates = df_candidates.sample(n = N)      # exploration
    df_candidates = df_candidates.drop('mask',axis = 1)
    df_interested[df_interested['movie_id'].isin(movie_ids)]['mask'] = 0
    df_rated_movies[df_rated_movies['movie_id'].isin(old_movies)]['mask'] = 1
```

```
    return df_candidates,movie_ids


def find_rewards(values,rewards, feedback_value):
    gamma = 0.2
    if feedback_value > 0 : value = 1
    else : value = -0.2
    values.append(value*gamma**t)
    rewards.append(sum(values[:t]))
    return values, rewards


def transition(P,old_state):                          #equivalent to the RNN␣
↪function. So it is the most complex and challenging function
    alpha = 0.6
    next_state = (P[P['rating'] > 0]['class'].value_counts()*alpha + old_state *␣
↪(1-alpha)).fillna(0)
    next_state = next_state / next_state.sum()
    next_state = round(next_state * N)
    if next_state.sum() < N :
        next_state = old_state

    return next_state

def restart():
    t = 0
    df_rated_movies['mask'] = 1
    rewards = []
    values = []
    feedback_values = [0]
    old_movies = []
    states = []

    initial_state = df_interested['class'].value_counts(normalize = True)*N   #␣
↪State Space ( discrete :-) ) [Out of 3 items , how many belong to each class␣
↪is each state. That is 4 non-negative integers add upto 3]. 20 ways are there.
    P , old_movies = generate(initial_state, old_movies)   # Action Space (␣
↪discrete :-) )
    states.append(initial_state)

    values , rewards = find_rewards(values , rewards, feedback_values[t])
    return rewards, values, feedback_values, old_movies, states, initial_state, P
```

[7]:
```
df_raw = analytics.DataFrame(data = [maths.arange(0,N+1)]).T
df_merge = analytics.merge(df_raw, df_raw, how = 'cross',suffixes=('_1','_2'))
df_merge = analytics.merge(df_merge, df_raw, how = 'cross',suffixes=('_x','_y'))
df_merge = analytics.merge(df_merge, df_raw, how = 'cross')
```

```
df_merge['sum'] = df_merge.sum(axis = 1)
df_all_states = df_merge[df_merge['sum'] == N].drop('sum',axis = 1).
↪reset_index(drop = True)
df_all_states.columns = list(maths.arange(N+1))
all_states = []
for i in range(len(df_all_states)) :
    all_states.append(df_all_states.iloc[i].astype('float64'))
```

5.1 Policy Iteration

[8]:
```
t = 0
rewards, values, feedback_values, old_movies, states, initial_state, P =␣
↪restart()
recommendations = []

print(initial_state)

print("Rewards :",rewards[-1])
```

```
class
2    1.293651
0    0.825397
1    0.746032
3    0.134921
Name: proportion, dtype: float64
Rewards : 0
```

[9]:
```
while t <= len(df_interested) :
    t = t + 1
    feedback_values = feedback(P,feedback_values,t)
    state = transition(P,states[t-1])
    states.append(state)
    P , old_movies = generate(state,old_movies)
    feedback_values = feedback(P,feedback_values,t)
    recommendations.append(P)
    values , rewards = find_rewards(values , rewards, feedback_values[t])


states_policy = states
rewards_policy = rewards
recommendations_policy = recommendations

print("Rewards :",rewards[-1])
```

```
Rewards : 0.049615999999999987
```

5.2 Value Iteration

[10]:
```
t = 0
rewards, values, feedback_values, old_movies, states, initial_state, P =␣
↪restart()
print(initial_state)
recommendations = []

print("Rewards :",rewards[-1])
```

```
class
2    1.293651
0    0.825397
1    0.746032
3    0.134921
Name: proportion, dtype: float64
Rewards : 0
```

[11]:
```
while t <= len(df_interested) :
    t = t + 1
    feedback_values = feedback(P,feedback_values,t)
    accumulate_temp_rewards = []

    for temp_state in all_states :
        temp_P , temp_old_movies = generate(temp_state,old_movies)
        temp_feedback_values = feedback(temp_P, feedback_values,t)
        if temp_feedback_values[t] > 0 : l = 1
        else : l = -0.2
        gamma = 0.2
        accumulate_temp_rewards.append(l * gamma ** t)

    state = all_states[maths.argmax(accumulate_temp_rewards)]
    states.append(state)
    P , old_movies = generate(state,old_movies)
    feedback_values = feedback(P,feedback_values,t)
    recommendations.append(P)
    values , rewards = find_rewards(values , rewards, feedback_values[t])

states_values = states
rewards_values = rewards
recommendations_values = recommendations

print("Rewards :",rewards[-1])
```

```
Rewards : 0.04999999901695999
```

References

[1] Liwei Huang, Mingsheng Fu, Fan Li, Hong Qu, Yangjun Liu, and Wenyu Chen. A deep reinforcement learning based long-term recommender system. *Knowledge-Based Systems*, 213:106706, 2021.