Documentation :

• Introduction:

The video recommendation system we are developing utilizes content-based filtering with the Jaccard similarity algorithm to provide personalized video recommendations to users. The system analyzes video content and user preferences to find similarities and suggest relevant videos based on their previous interactions. To demonstrate and test the system, we have generated a list of 300 random video titles. These titles will be stored in a Java Script Object Notation (JSON) format, which will be processed by the system. It's important to note that the generated list of titles is not static, as users can upload additional titles during the runtime of the simulation. After generating the list of video titles, we will tokenize them into tags using simple tokenization algorithms provided by the NLTK (Natural Language Processing Toolkit) library. These tags will be stored alongside the videos, allowing us to extract meaningful attributes from the video content. The final JSON format for each video entry will include the video title and its corresponding tags. This information will be used in the content-based filtering process to calculate the Jaccard similarity between videos and recommend similar videos based on user preferences and past interactions. Overall, our objective is to create a recommendation system that offers personalized and engaging video suggestions by leveraging content-based filtering and the Jaccard similarity algorithm.

• Architecture:

my_exceptions.py :

Class ErrorMessages:

Contains all error messages.

Class SystemMessages:

Contains all system messages.

Class EmptyError, UnicityError, LoggingError:

Inherits from the base Exception class.

my_datastructures.py :

Class Array:

Attributes:

- _type : Any
- _size : int
- _capacity: int
- _Array : Static Array(ctypes)

Methods :

__getitem__(k : int) -> Any : Returns element an index k.

__len__( ) -> int : Returns the size of the array.

__setitem__(k : int , item) -> None : Sets an element at index k.

_make_array(capacity : int) -> ctypes.py_object : Return new array with a new capacity .

pop(index : Int=0) -> Any : Remove the value at index and returns it .

resize(capacity : int) -> None : Resize internal array to a new capacity .

Class heap:

Attributes:

- _type : Any
- _content : array

Methods :

__len__( ) -> int :Returns the number of nodes the tree has .

_swap(i: int , j: int) :Swaps two elements in the tree ..

pop ( ) -> Any :This method removes the root element from the heap and returns it .

push (item : Any) -> None :This method adds an element to the heap .

Class stack:

Attributes:

- size : int
- head : node

Subclass:

_node: attributes: element: Any, next: _node .

Methods:

__len__( ) -> None : Returns the size of the stack .

is_empty( ) -> bool : Returns True if the stack is empty ,False otherwise.

peek( ) -> Any : Returns the top of the stack.

pop( ) -> Any : Removes and returns the element at the top of the stack .

Returns :the element at the top of the stack .

push(element: Any) -> None : Add new element onto the top of the stack .

my_classes.py :

Class session:

Attributes :

- current_user : User

subclass :

_entry: attributes : id: int, score: float.

Methods :

- __lt__(other) :Compares the current instance with another '_entry ' object based on their scores .
- __eq__(other) :Compares the current instance with another '_entry' object based on their scores .

Methods :

create_account (username: str , password: str) -> User : This method creates a user.

delete_account(password: str) -> None : This method deletes the user from the system and all his videos and calls logout.

get_id( ) -> int : Generates an ID .

get_session( ) -> Session : Simple getter for the _instance .

load ( ) -> None : Loads the dataset into the system.

login(id: int , password: str) -> User : This method will authenticate as a user to be simulated upon.

logout( ) -> None : Logs out the user .

recommend(Query: optinal [str] = None) -> Heap : This method will authenticate as a user to be simulated upon.

save( ) -> None : Saves the system into the dataset.

search(Query: str) ->Stack : This returns a stack containing videos with similar titles.

Class User:

Attributes :

- name : str
- id :int ,video_count : int
- subscriber_count : int
- tags : list[str]
- uploaded_videos : list[int]
- subscribed_to : list[int]
- watch_history : Stack[int]
- Private: __password: str

Methods :

__repr__( ) -> str : This method returns all the information relative to the user in a string.

dislike(video_ID: int) -> None : This method adds dislikes to the video.

like (video_ID: int) -> None : This method adds likes to the video.

subscribe(user_ID: int) -> None : This method subscribes to a user with the specific ID.

unsubscribe(user_ID: int) -> None : This method unsubscribes to a user with the specific ID.

upload (title: str) -> my_classes.Video :This method adds a video with a specific title to the system.

unupload(video_ID: int) -> None : This method removes the video with a specific ID.

watch_video(video_ID: int) -> None :This method simulates the action of viewing the video.

Class Video:

Attributes :

- name : str
- owner_id : int
- id : int

- likes : int
- dislikes : int
- views : int
- tags : Array[str].

Methods :

Is_author(user_ID: int) -> bool : This method checks if the user owns this video.

__repr__( ) -> str : This method returns all the information relative to the video in a string.

Functions:

my_classes.py :

Handle_login_error(func) -> callable :

- Decorator function that handles login errors by raising a LoggingError if the current user is None.

Recommendation_core.py :

extract_keywords(sentence) -> list [str]:

- Takes a sentence and returns a list of the main keywords

jacard_calculation(set1: set, set2: set) -> float:

- Counts the similarity percentage between two sets

ratio_additivity(scores: int, likes: int, dislikes: int, views: int) -> float:

- Takes into account the like/dislike ratio and views

Classes and Interactions in a Simulated Video Recommendation System:

Video class: Represents a video object with attributes such as name, owner ID, ID, likes, dislikes, views, and tags. Utilizes arrays to store the video attributes. Provides methods to check if a user owns the video and manipulate its attributes.

User class: Represents a user object with attributes such as name, ID, video count, subscriber count, uploaded videos, subscribed users, watch history, and tags. Utilizes arrays to store the user attributes, such as uploaded videos, subscribed users, and watch history. Provides methods to upload, delete, like, dislike, subscribe, unsubscribe, and watch videos. Utilizes stacks to manage the watch history, allowing users to track their recently watched videos.

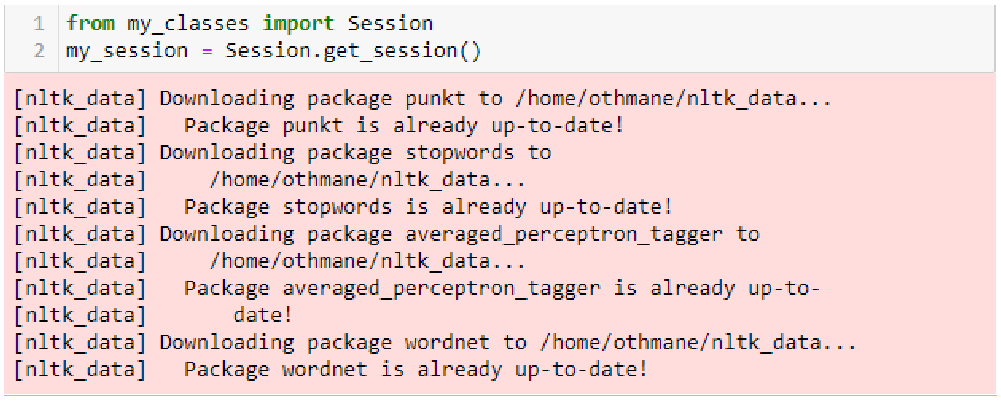
Session class: Implements the Singleton pattern to manage user sessions. Utilizes a heap data structure to recommend videos based on user preferences. Maintains a current user and provides methods for login, logout, video recommendations, and searching videos. Utilizes arrays to store the user sessions and their associated data.

These classes leverage arrays, heaps, and stacks to manage and manipulate video and user data efficiently.

The provided code defines classes for a simulated video recommendation system. The main classes are Video and User. The Video class represents a video and stores information like its name, owner, likes, dislikes, views, and tags. The User class represents a user and keeps track of their details such as name, ID, uploaded videos, subscribers, and watch history. Users can perform actions like uploading videos, liking/disliking videos, subscribing to other users, and watching videos. The Session class manages user sessions and system operations. It handles tasks like logging in/out, recommending videos based on user preferences, searching for videos, and managing user accounts. It also handles loading and saving data from external files. Overall, these classes work together to create a video recommendation system where users can interact with videos and perform various actions, while the Session class controls the user sessions and manages system operations.

• Approach (How it works?):

The Session class provides a user session for interacting with the video platform system. In the provided code, a session is created by calling Session.get_session(), which ensures that only one instance of the Session class exists.



```python
from my_classes import Session
my_session = Session.get_session()
```

```
[nltk_data] Downloading package punkt to /home/othmane/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/othmane/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/othmane/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /home/othmane/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Then, the user logs in with the ID 344 and password "459" by calling my_session.login(344, "459"). If the login is successful, the user object is returned and stored in my_user.

If the user object does not have an account, they can use the create_account method to provide a username and password, and then proceed to log in.

```
#my_user = my_session.create_account("Othmane", "459")
my_user = my_session.login(344, "459")
```

Next, a recommendation heap is obtained by calling my_session.recommend(). The while loop iterates over the heap to print the details of the recommended videos. The loop runs until either the heap is empty or 10 videos have been printed.

```
heap = my_session.recommend()

i = 0
while len(heap) != 0 and i < 10:
    vid_id = heap.pop().id
    print(my_session.registered_videos[vid_id], vid_id)
    i += 1
```

After that, the user watches two specific videos using the my_user.watch_video() method. The video with ID 91 ("10 Tips for Starting a Successful YouTube Channel") and video with ID 339 ("Exploring Cultural Festivals: Celebrating Diversity") are watched. Then, another recommendation heap is generated by calling my_session.recommend().

```
my_user.watch_video(91)
my_user.watch_video(339)
```

Finally, a search is performed by calling my_session.search("Innovation"). The search returns a stack of videos with similar titles to "Innovation". The while loop iterates over the stack, printing the details of the videos along with their IDs and scores. The loop continues until the stack is empty.

```python
# while len(my_user.tags) != 0: my_user.tags.pop()  #Clear TAGS
stck = my_session.search("Innovation")

while len(stck) != 0:
    entry_ = stck.pop()
    vid_id = entry_.id
    vid_score = entry_.score
    print(my_session.registered_videos[vid_id], vid_id, vid_score)
```

Finally, the session's state is saved by calling the save() method, ensuring that the registered videos and users are persisted in the dataset.

```python
session.save()
```

Overall, the Session class provides a comprehensive set of functionalities for user interactions with the video platform system, including login, recommendation, search, video watching, and saving the system state. Users can leverage the like, dislike, and watch methods to influence recommendation scores and personalize their video recommendations based on their preferences and interactions.