Implementing Video Recommendation System

Objective: The objective of this project is to develop a video recommendation system that utilizes content-based filtering with the Jaccard similarity algorithm. The system aims to provide personalized video recommendations to users based on their preferences and the similarity between the content attributes of the videos they have engaged with. By employing the Jaccard similarity, we aim to enhance the accuracy and effectiveness of the recommendation system in suggesting relevant and engaging videos to users.

Introduction

Our recommendation system delivers personalized video recommendations to users based on their previous interactions with similar videos. We use a content-based filtering approach, analysing video content and user preferences to find similarities and suggest relevant items.

By prioritizing the content, itself, we don't rely on similar user behaviours, resulting in accurate and customized recommendations. Our aim is to enhance the user experience by offering engaging and personalized video suggestions that match users' preferences and past interactions.

Dataset

Source: Created.
Generating a list of 300 random videos titles for testing and demonstration purposes.
The generated list of titles will be stored in a Java Script Object Notation format, to be processed by the system. The generated list isn't static as the user could upload titles during the runtime of the simulation.
After the generation of the list, the titles will be broken into TAGS using some simple tokenization algorithms contained in the NLTK (NATURAL LANGUAGE PROCESSING TOOLKIT) library.
The TAGS will be stored alongside the videos.

The final JSON format will be as follows:

videos.json

```
{
        Video_ID: (title, like_count, dislike_count, views,
        OwnerID, [ *Tags  ]),
}
```

Data Structures Design (Dataclasses)

In this part we will discuss the dataclasses used for the project.

1. User Profiles:

Implement a "User" class to represent user profiles with the following attributes and methods:

- name: str
- ID: int | The user's unique ID.
- uploaded_videos: List[int] | A list of video_ID's of the user's owned videos.
- subscribed_to: List[int] | A list of user_ID's the user has subscribed to.
- watch_history: Stack[int] | A stack containing video_ID's the user has recently watched.
- video_count: int | A counter to how many videos the user owns.
- subscriber_count: int| A counter of how many other users has subscribed to the current user.
- tags (interests): List[str] | A list containing the most frequently watched tags by the user.
- __password: str | User's pasword.
- __repr__ () → str | This method returns all the information relative to the user in a string.
- upload (title: str) → 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.
- like (video_ID: int) →None | This method adds likes to the video.
- dislike (video_ID: int) →None | This method adds dislikes 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.

2. Video structure:

Implement a "Video" class to represent individual videos with the following attributes and methods:

- name: str | The video's title.
- ID: int | The video's unique ID.
- likes: int | The video's like count.
- dislikes: int | The video's dislike count.
- views: int | The video's views count.
- owner_id: int | The user_ID of how owns the video.
- tags: array[str] | An array containing the tags of the video.
- __repr__() → str | This method returns all the information relative to the video in a string.
- is_author (user_ID: int) → bool | this method checks the user.

3. Class node:

This class will contain the following attributes:

- data: Any | Held information.
- next: *node | A pointer to the next node

4. Recommendation output:

The output will be represented as a binary minheap stored in a dynamic array; this choice was made since the recommendation process is a minimal score based. That means that the output must be stored in the heap implementation of the priority queue, this decision was made to efficiently use space and time complexities.

5. FILE storage:

There will be two JSON files:

- The videos JSON: Will contain information relative to all registered videos.
- The users JSON: Will contain information relative to all registered users.

Data Structures Design (Classes):

Class Heap:

This class will contain the following attributes and methods:

- _type: Any (THIS TYPE MUST SUPPORT >, < and = OPERATIONS) | The heap's contained type of data.
- _content: array | The heap's content.
- __len__() → int | Returns the number of nodes the tree has.
- _swap(i: int , j: int) | Swaps two elements in the tree.
- push (item: Any) → None | This method adds an element to the heap.
- pop () → Any | This method removes the root element from the heap and returns it.
- heapify () → None | This method ensures the heap invariant.

Class Array:

This class will contain the following attributes and methods:

- _type: Any | The array's contained type of data.
- _capacity: int | The array's capacity.
- _size: int | The array's size.
- _Array: Static Array (ctypes) | The static array that holds the information for this dynamic array class.
- __len__ () → int| Returns the size of the array.
- __getitem__ (key: int) → Any | Based on a key this method will return the element located at that key in the array.
- __setitem__ (key: int, item) → None| Based on a given key, this method will overwrite the element located at that key in the array.

- _make_array (capacity: int) → ctypes.py_object | Reutrn 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 Stack:

This class will contain the following attributes and methods:

- size: int | The stack's size.
- head: node | The stack's head node.
- Subclass: _node Parameters : (element , next) →None.
- is_empty () → bool | This method returns whether the stack is empty or not.
- push (element: Any) → None | This method adds an element to the top of the stack.
- pop () → Any | This method removes the top element from the stack and returns it.
- peek() →Any | Add new element onto the top of the stack.
- __len__( ) → None | Returns the size of the stack.

Class Session:

This class will contain the following attributes and methods (this class is responsible for the runtime):

- current_user: User | The current logged in user.
- Subclass: _entry parameters (id: int ,score: float) → None.
- login (id: int , password: str) → User | This method will authenticate as a User to be simulated upon.
- logout () → None |logout the user and returns it.
- recommend (Query : optinal[str]=None) → Heap | This method returns the heap containing the recommendations.
- search (query: str) → Stack | This returns a stack containing videos with similar titles.
- save ( ) → None | Saves the system into the dataset.
- create_account (username: str, password: str) → User | This method creates a user and returns it.
- delete_account (password: str) → None | This method deletes the logged in user and his owned videos.

Class ErrorMessages:

Contains all error messages.

Class SystemMessages:

Contains all system messages.

Final Outcome

The outcome will be a list of recommended videos for the user. The minimum recommended videos should be the top 10 elements of the heap, while the maximum should be the top 40 elements of the heap.
We can test this process using the following implementations:

1. Application:

At the start the user can log in with his username and a password.

Once the user is logged in, he will be provided with options such as search for videos, upload own video, view watch history, view recommendations, and subscribe to channels.

2. Simulation:

To validate the effectiveness and performance of our recommendation system, we will conduct a final simulation. By testing the system using sample user profiles and evaluating the quality of the recommendations provided, we can assess its capabilities and refine it further. This simulation allows us to make data-driven improvements, ensuring that our video recommendation system delivers accurate and engaging content suggestions to users.