Master's Project
Title : Prototypage d'un système de recommandations de vidéos à la demande
Master of Computer Science – MFLS

Noozy AI

Development of a recommender system for video-on-demand platform

---

Asif Ahmed
asif.ahmed3@etu.univ-lorraine.fr

Team: BIRD
Loria, Nancy

Team Members
Olfa MESSAOUD
Supervisor: Prof. Azim ROUSSANALY

June 29, 2021

Abstract

Recommender algorithms can guide users in a personalized way to interesting objects in a large space of possible options. The necessity of recommendations is increasing on cultural or entertainment and media industries, where the number of products is continuously increasing. Cultural and media platforms and digital markets are getting heavily benefitted from implementing, maintaining, and improving their recommender system. The process on how they retrieve cultural products like music, books, movies, news and enable easy access to the users can have a structural impact on how markets operate alongside how consuming trends change. The prospect of this project is to engineer a complete recommender system for noozy.tv, a new video-on-demand platform dedicated for the viewers of Grand Est region in France. The aim is to develop a framework maintaining standard and modern software development methodologies and tools to ensure seamless service, research scope on real data and diversity, evaluation, and delivering a platform for further improvement in system.

Résumé

Les algorithmes de recommandation peuvent guider les utilisateurs de manière personnalisée vers des objets intéressants dans un large espace d'options possibles. La nécessité de recommandations augmente sur les industries culturelles ou du divertissement et des médias, où le nombre de produits ne cesse d'augmenter. Les plateformes culturelles et médiatiques et les marchés numériques bénéficient grandement de la mise en œuvre, de la maintenance et de l'amélioration de leur système de recommandation. Le processus sur la façon dont ils récupèrent les produits culturels comme la musique, les livres, les films, les actualités et permettent un accès facile aux utilisateurs peut avoir un impact structurel sur le fonctionnement des marchés parallèlement à l'évolution des tendances de consommation. La perspective de ce projet est de concevoir un système de recommandation complet pour noozy.tv, une nouvelle plateforme de vidéo à la demande dédiée aux téléspectateurs de la région Grand Est en France. L'objectif est de développer un cadre maintenant des méthodologies et des outils de développement de logiciels standard et modernes pour assurer un service transparent, une portée de recherche sur les données réelles et la diversité, l'évaluation et la fourniture d'une plate-forme pour une amélioration supplémentaire du système.

Acknowledgements

I have learned a lot while working on this project. With modern and effective tools and methodologies, my supervisor introduced me dependable software engineering and development standard for placing a base application for a growing engineering team and let me be a part of this project. I must express my utmost gratitude to Prof. Azim ROUSSANALY for his trust, encouragement, and guidance. I am also thankful to my senior member in the team, Olfa MESSAOUD for her works and advice.

Working on this project day and night was joyful for me. Although, due to the pandemic I had to work from home, but my team gave me complete flexibility of working and communicating with them.

I would like to thank Prof. Didier GALMICHE for his welcoming support and his way of motivations which was extremely favorable for me. I also want to thank the jury for reading and evaluating my modest works and learnings here.

And finally, I am grateful to Erasmus Mundus for offering me and many others such opportunity to experience life and pursue higher studies in a new and dynamic environment.

Table of Contents

List of Tables

List of figures

List of Appendices

1. Introduction

Artificial Intelligence (AI) has emerged as a technical scenario of underlying modern-day real-life solutions to increasingly important tasks in daily life and work. AI solutions are typically come in the form of software as a service (SaaS). Which also can be called Intelligence Software [intelli soft]. For any type of internet service, marketing, entertainment and commerce, recommender system has become extensively used AI system that are boosting revenues and maintaining digital consuming trends. For any web media portal such as – YouTube, Netflix, Canal+, Sky, Amazon Prime, Spotify etc., their own recommendation algorithms play a huge role for consumer satisfaction and marketing.

The requirement of this project is to develop the prototype of the recommender intelligence software Noozy AI for the web tv platform noozy.tv. The main goal of this project is to –

- design the architecture of the recommender system for easy integration to servers for a seamless communication with platform and evaluation
- develop a dependable and sustainable software, maintaining standard specification and coding principles
- ensuring a framework with test facility, choice of algorithm models with different parameters
- research scope with real data for case study on diversity and efficiency of the system
- satisfying *noozy.tv*'s requirements and place an ideal application for easy and further improvements

The main study of this article is as follows,
This chapter emphasize on video-on-demand platform and its recommender system. Chapter-2 defines the core components, strategies, process, and the architecture of Noozy AI and its engineering steps. In chapter 3, the design and development of the framework is described with cumulative diagrams and process flows. It can be taken as a technical specification and engineering guide to this prototype platform. In chapter 4, we discuss briefly about the reliability and the standards of programming for this platform. Chapter-5 presents the implementation and study of Content-Based algorithm integrated on this platform. After that, the evaluation process of such system is on chapter 6. The analysis and evaluation of the prior

case study is discussed here. The following chapters states about the ideas of hybrid algorithm from the case study and prospect of research on diversity. And conclude the report with on-the-hand tasks of future developments on this platform.

1.1. VoD

Video on demand platform is a media distribution platform that allows users to access video through web application. Broadcasting media over-the-air was a common media of entertainment for everybody, once. Now, plethora of web tv platform, podcast and IP TV has taken over the traditional television concept and consumers began to lean towards this new mode of consumption through internet. We can watch new and our favorite old TV programs, movies along with live broadcasted shows and news in our computers, mobiles, tablet devices and smart TVs. Peer-to-peer (P2P) file sharing systems enabled the distribution of media content over the web, without the linear cost associated with streaming media. The viewers of this generation are not spending their time to wait in front of a television set to watch a show. Cultural and media platforms such as Netflix, Amazon Prime, Sky, Canal+, Disney+ etc. are dominating entertainment broadcasting world with digital streaming.

Along with the benefit of not waiting for our favorite shows or important news on TV, we got the idea of choosing what to watch, when to watch and liberty to watch them from any place. This domain has become enormous in terms of technologies, platforms, and data. In our daily life, we rely on recommendations from other people, newspapers, books, reviews, guides and informercials. Then, use of recommender system has made this dilemma of 'choosing' much lesser for the viewers.

1.2. Recommender System

The ongoing rapid expansion of the digital media market has greatly increased the necessity of better and constantly upgrading recommender systems. Digital platforms are increasing their revenue in a huge amount with improving their recommender algorithms and service. The first noted recommender system *Tapestry* [1] was developed with collaborative filtering methods to filter out mails, but that was manual in more sense. Where, it required a qualified human annotation to get computed filtered mails. With automated filtering using various fields of data and many different algorithms to compute, filtering and delivering of content is

now known as recommender system. The idea of collaborative filtering is still the driving force of behind many successful media platform and market.

A recommender system uses inputs to predict potential interests of viewers. These inputs can be of two types – *Knowledge* and *Data*. In our system we are considering video metadata (set by noozy tv) as knowledge and user's information and interaction with the platform is taken as data. While computing, the many different recommendation algorithms use these inputs and its parameters and gives recommendation for the viewers.

1.2.1. Basic recommendation

The most basic recommenders can be as followed,

i. *Random Videos*: In which section, viewer will be recommended N number of videos randomly, sometimes with considering different metadata parameter.
ii. *Most-Popular Videos*: In this section, viewers will be given the top N most viewed video lists in a sorted order.
iii. *Most-Recent Videos*: With this, users will be shown the top N most recently uploaded videos.
iv. *Rating-based Video Recommendation*: This can be seen as aggregated opinion approach of recommending videos. Based on rating or review on contents, it will give top N top rated videos.

(N is a generic number)

These recommenders are the simplest in type and does not consider user's personal tastes nor their *data*, rather than completely depending on simply metadata information or *knowledge*. But, considering user's ease, these algorithms will deliver only user's unseen videos with these simple recommendations, if the user is authorized or a member of the platform.

1.2.2. Advanced Recommendation

For starters, our team worked on various types of Collaborative filtering and Content Based filtering as a bit more advanced type of recommendation. This type can be named as *Personalized Recommendation.* Where the algorithms will consider taking user's personal information such as, age, gender, location, interests, watched videos etc. Alongside user's data, content's information will be used to compute recommendation.

Collaborative filtering bases on the appreciation of a set of users on items, to make either User Based [6] recommendations or Item Based recommendations [7]. Content Based filtering recommends similar videos to the contents of a specific video (Item Based) or videos appreciated or watched by the current user (User Based) [8]. And then comes *Hybrid filtering*, which combines approaches to decrease the issues resulted from both approaches mentioned above.

Apart from algorithms and their usefulness of recommendations, all AI or intelligence software systems need to provide other services, such as – performance, security, reliability, maintainability, and dependability. The right to evaluate the recommendations by the platform is also a key part for any evaluation of algorithms here.

2. Noozy AI

- Noozy TV is a video on demand platform, a part of CEI project consisting of developing a video streaming platform for the Grand Est Region of France. This platform is developed by *Kardham Digital* [9] with the contributors of *Alsace20*, *Canal32*, *Moselle TV* and *Vosges TV*. Its main purpose is to focus on the regional audio-visual productions that were limited to local visualizations and valorise the regional culture by diffusing content for a wide range of users.

Noozy TV also integrates the intelligence system to analyse users and know more about their preferences so to offer them personalized contents. The major potential of every streaming platform is the quality of the integrated recommender system. Team - BIRD of Loria, Nancy is working on the development and managing the system naming *Noozy AI*.

Intelligence software is naturally compliant to software engineering issues such as dependability [10] including reliability [11], security, evaluation etc. In an agile software development environment, the architecture of the system plays a key part for the start of a sustainable engineering project. We had to take requirements of making the process flow understandable, selecting integration methods, set a secured environment and keep it lightweight in computational cost. And to keep the prospect of future possibilities and

requirement to come, the development is done by maintaining coding standard and principles for programming, managing data asynchronously, and keeping data security.

Noozy AI is backend process of gathering knowledge and data from source, computing various types of recommendations based on those knowledge and data, storing them in a repository and serving those as per request from *Noozy TV* platform. It is a model framework on which, various recommender algorithm will be integrated. And then, it will be a platform for research on hybrid recommendation and diversity with real data considering local viewers.

The system will take requests from the web tv platform and responds with asked recommendation. It will not be a real time computation. This type of extensive data and computation-oriented services are designed with repository to store computations in advance. And being computed over a considerable amount of time, repeatedly.

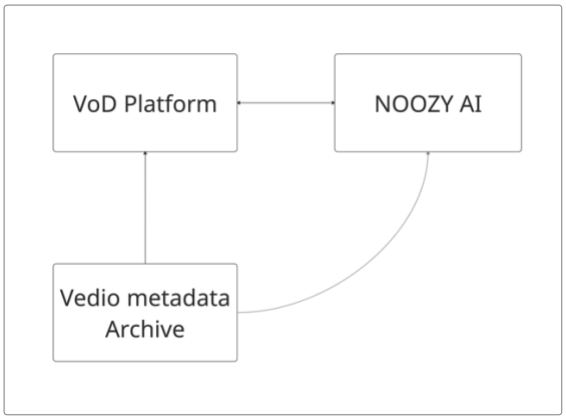

Figure 2.0.1. VoD platform and Recommender

All the projects of the processes on collecting needed knowledge and data, clean, analyze and compute various kinds of recommendation, storing them and serving request from the platform combined is *Noozy AI.* The stack of the project modules itself is the recommender system.

The major parts of the complete process are – knowledge gather, data collection and repository, compute with recommender AI, repository the recommendations and communicate with the portal. All of these are by itself communication and computation costly applications. Therefore, the system is designed considering micro-service architecture. By which, each application gets to lightweight micro-services, communicating each other or with common resources. The main micro-service applications developed are – metadata harvest and repository, LRS to store and serve experience API data, algorithm modules, recommendation repository, and recommender API gateway.

2.1. Micro-service architecture

Migrating monolithic architectures to cloud-native architectures like microservices creates many advantages such as flexibility to adapt to the technological differences and independent resource management for different system components. A monolith software is an application whose modules cannot be executed independently. This makes monoliths difficult to use in distributed systems without specific frameworks. Also, monoliths limit scalability and large-size applications are difficult to maintain or extend due to their complexity. For such AI software developing, monolith application is not preferable [12].

A microservice [13] is a cohesive, independent process interacting between themselves or sharing same the resources. A microservice architecture based software is a distributed application where all its modules are microservices. It enables easy integration to cloud servers. It offers maintainability for finding and fixing errors without stopping the whole system.

2.2. Docker Container

The popular emerging technology Docker combines several areas from systems research and development - such as operating system virtualization, cross-platform portability, modular re-usable elements, versioning, and integration. A docker container is a running instance of an image. [14] Sharing the Linux kernel makes Docker much more lightweight and higher performing than complete virtual machines.

The micro-services in Noozy AI are dockerized by keeping a template project structure for easy install and updates. Also, all the processes are modular and independent. So, study, research and improving on algorithms will be much easier to implement, integrate and testing.

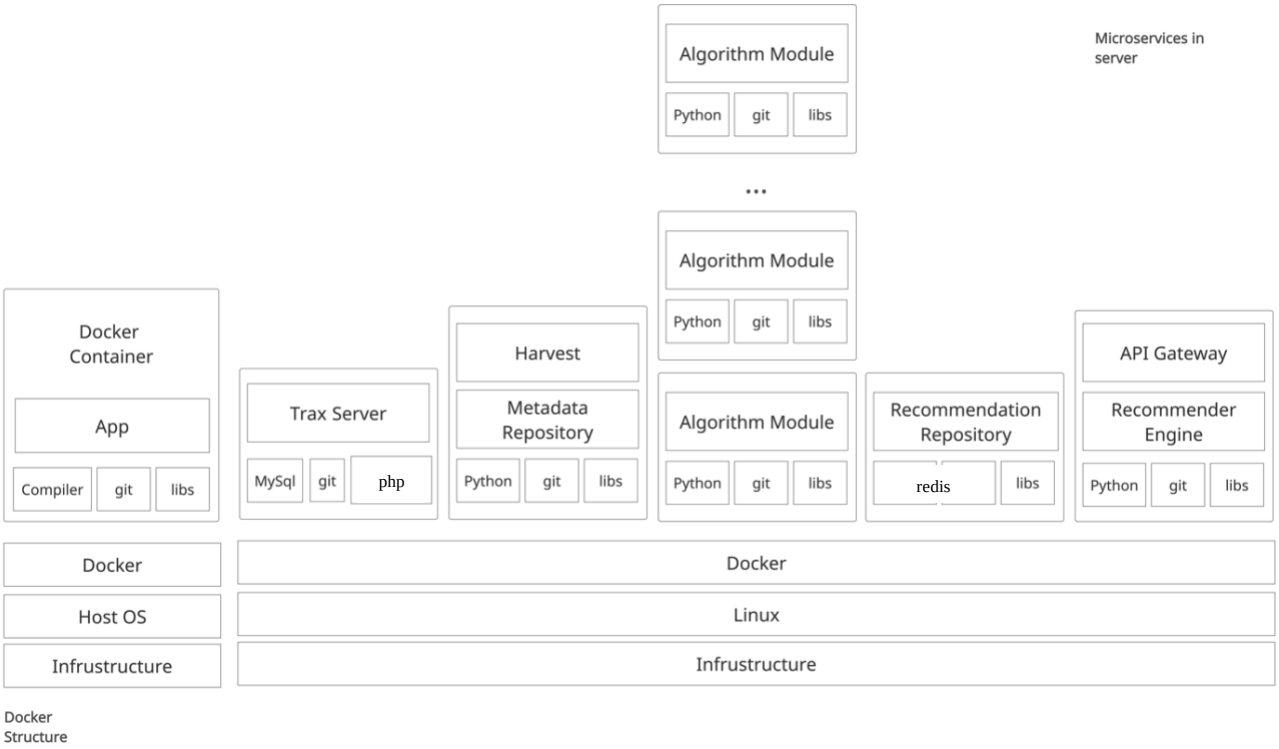

Figure 2.0.2. Structure of Noozy micro-service docker containers

All the applications of the system are installed as containers on the servers (test and final). Where, docker is installed over the host operating system.

2.3. Daemon

Among all the processes, the *harvest* and all the *algorithm modules* will act as daemon[15] and will continuously run over a scheduled period. Trax server and recommender engine are already daemon running to communicate in a request-response manner. The other computation and storing related applications are implemented as daemon where the process gets initiated over a period to keep data and recommendation updated, as *harvest* process. The recommender engine acts as a server to respond to recommendation requests.

2.4. Repository and Result store

Redis[16] is an open source (BSD licensed), in-memory data structure store which is used as a database, cache or message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries. It combines the best of in-memory, no-schema design with optimized data structures and versatile modules that adapt to your data needs. The result is the most adaptable, high performance, multi-purpose database, that scales easily like a simple

key-value data store but delivers sophisticated functionality with very little complexity. Redis offers faster and efficient query functionalities such as GET, SET, DEL etc., with complexity O(1). Unlike other key-value stores, Redis allows access to discrete elements within objects, eliminating serialization/deserialization and processing boilerplates. Which places a crucial part of flexibility for intelligence systems, IoT systems and highly computation-oriented software systems.

To note, redis in general does not have JSON data structure as its type. JSON is a lightweight data-interchange model and popular and standardized for its easy human understandability. [17]. To store data in json format, RedisJSON[18] is used. It is a redis module that implements json as native data type for storing values and transmission in server-client requests.

2.5. Environment

The architecture is designed keeping ubuntu server as platform. It is programmed mainly in Python programming, and all the projects are developed in IntelliJ Idea IDE [19]. All the projects are developed in their virtual environments mentioning all the necessary libraries and modules required, so that versioning of libraries and tools do not affect them. The API gateway will server requests in openAPI specification. For research study Jupyter Notebook [20] is used.

2.6. Version Control System (VCS)

In software engineering, version control is a class of systems responsible for managing changes to computer programs or other collections of information. Version control is a component of software configuration management. We used git as the VCS of the modules [21] and it enables easy integration and continuous update functionality, Loria's private gitlab[22] is used to repository the projects.

2.7. Integration

All the processes are developed based on testing on *Test Server* and integrated for production on the *Final Server*. The abstract of the integration process is fetching project source code from git repository and build docker container of that project to start a daemon in servers. With

easy logging functionality and update commands, we keep the process of development running first on Test Server and then deploy to Final Server.

Table 2.1. Server Specification

	Test Server	Final Server
Architecture	x86_64	x86_64
CPU Memory	2.4 GHz	2.4 GHz
No. of CPU	1	2
Memory	1.9 gb	15 gb
Host OS	Debian GNU/Linux 10	Debian GNU/Linux 10

2.8. Testing

Software testing is one of the key-part of dependable software engineering and producing a reliable software. Not only all the recommendation has to meet the expectation of the editors of the platform at first, but also, the application must maintain seamless, and error free process run and communication. Thus, creating test cases and matching functional and structural results according to them is very important. There are many methods for automatic testing technique and manual test options.

Traditionally, software testing can be classified into – Black box and White box testing

2.8.1. Black Box Testing

Black box testing is the functional testing of a program. In our case, all the micro-services can be black boxes. The software tester does not know the internal mechanisms of the black boxes. They focus on the output generated by the response to selected input scenario and parameters. [23]

2.8.2. White Box Testing

On the other hand, white box testing is also called as structural testing or glass box testing. It is technique of designing test cases based on the information derived from source code [24]. The white box tester (most often the developer of the code) knows what the code looks like and writes test cases by executing methods with certain parameters. White box testing concerns the internal mechanism of a system. It mainly focusses on the control flow or data flow of a program [25].

White-box and black-box testing are considered corresponding to each other. Many researchers state that, to test software more correctly, it is essential to cover both specification and code actions. In all cases, for any recommender software's acceptability not only depends on the quality of recommendations, but also its reliability depends on all the other attributes of a service e.g., computation time, accuracy, network usage, easy process and continuity.

2.9. Licensing

The GNU General Public License (gnu gpl v3)[26] is a free, copyright license for software and other kinds of works. Everyone is permitted to copy and distribute this license document, but changing it is not allowed. It is constituted by Copyright © 2007 Free Software Foundation, Inc. [27]. This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions. The team accepted the Noozy AI's license to be of GNU lgpl v3.

2.10. Documentation

Documentation is a very important part of any software development process. No matter how good the software is, if the documentation is not good, no one will use it. And without a proper documentation, the development will not sustain.

Thus, along with the development of the service, proper and standardized documentation of the programmed methods, modules, projects, and APIs are to be written. This is generated with PEP 257 docstring conversion. The documentation is converted from Google style python docstring in modules, classes and methods by Sphinx tool [28] of IntelliJ IDE.

```
def method(param1, param2) -> Bool:
    """
    Example function with types documented in the docstring.

    Args:
        param1 (int): The first parameter.
        param2 (str): The second parameter.

    Returns:
        bool: The return value. True for success, False otherwise.

    """
```

Appendix 2-A. Google style python docstring

For every project, there is a README.md file documenting the installation, starting, stopping, update and other process. And it covers brief idea of the specification. Also, the programmed codes are easy to read, with comments and annotation on each step. It is essential for the team development.

3. Development

The development of Noozy AI starts from a real-time recommender application communicating with noozy.tv via RestfulAPI service. For each request of recommendation, it started for a computation by collecting data directly from the VoD platform. Which was a huge bottleneck considering video portal web application. The user experience data gets collected from Trax server each time. And from those input, recommendations were generated for each request. Apart from all the data related issues, the main issue was computational and network offload.

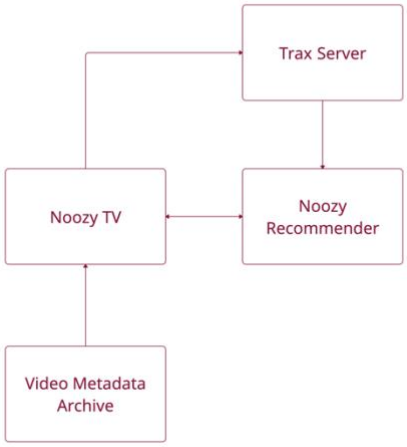

Figure 3.1. Noozy Recommender

For a growing number of users and contents in a consumer service, it requires a repository-based approach of solution. And the whole data flow of the system got much easier and reliable for engineering by using micros-service architecture and lightweight open-source tools such as Redis, git, docker etc.

All the components in Noozy AI system, is modular, independent and can be separately tested and updated. Thus, we develop using the discussed architecture for the whole system.

3.1. Design

The design of this framework starts with the main components – harvest metadata, collect xAPI statements, process them through algorithm, get and repository recommendations, and then serve to the portal. All of these are detailed in the later sub-sections. All the major components and any new component to come in future, are micro-services. Where, each of them are communicating either between themselves or sharing common resources. As the

process is scheduled, in each schedule the process starts with fetching already gathered knowledge and input data and starts recommending for a whole session. And then it store those in a readable and easily maintainable structure.

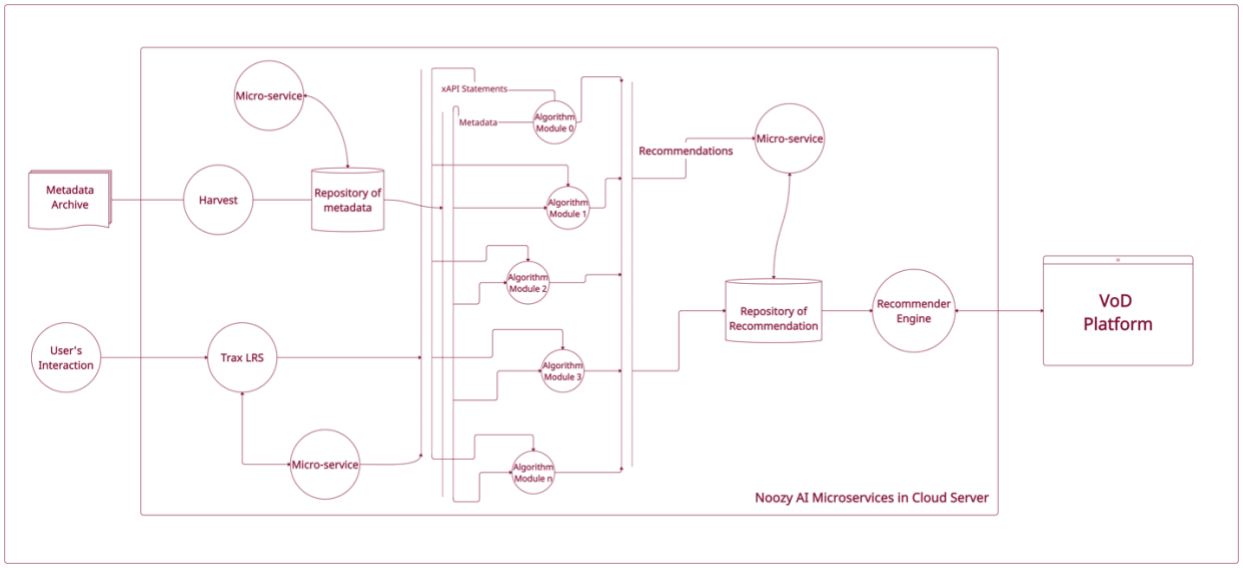

Figure 3.2. Noozy AI Microservices

3.2. Metadata Harvest

Metadata contains all the necessary information of all the contents on the portal. To start working, all the video metadata in the Noozy TV platform's archive gets collected, cleaned, structured and stored in a repository. The process is called *Harvest*. This is scheduled as per need and it keeps maintained repository, balanced with disclosing redundant communication load with the platform's server.

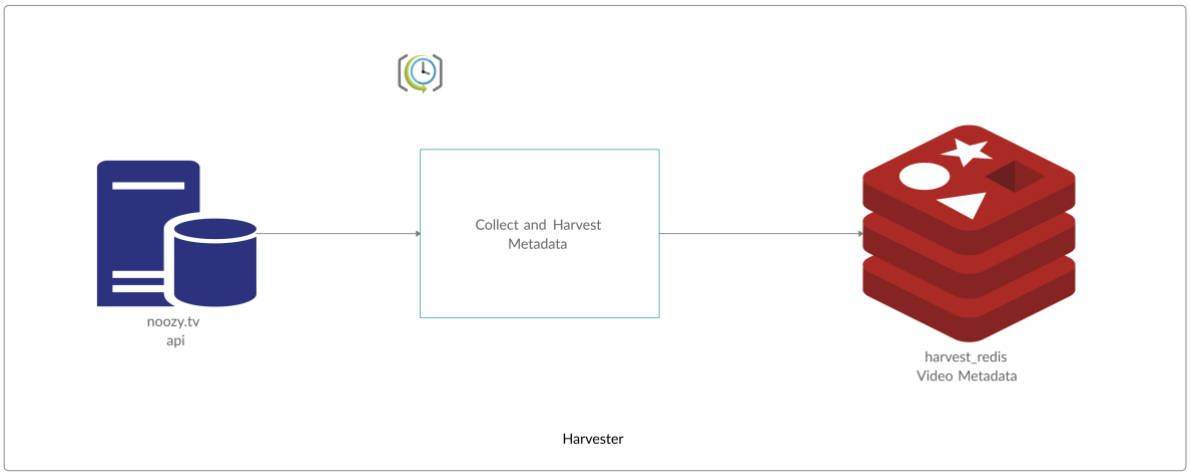

Figure 3.3 Metadata Harvest

To get a standard and all the core elements of a content, Dublin Core standard is followed for metadata structuring. It is a style of metadata that draws on multiple Resource Description Framework (RDF) [29] vocabularies, packaged and constrained in Dublin Core application profiles. The Dublin Core™ Metadata Element Set is a vocabulary of fifteen properties for use in resource description. [30]

Table 3.1. Dublin Core Metadata Elements

Field	Description of Value
Contributor	An entity responsible for making contributions to the resource."
Coverage	The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant.
Creator	An entity primarily responsible for making the resource.
Date	A point or period of time associated with an event in the lifecycle of the resource.
Description	An account of the resource.
Format	The file format, physical medium, or dimensions of the resource.
Identifier	An unambiguous reference to the resource within a given context.
Language	A language of the resource
Publisher	An entity responsible for making the resource available
Relation	A related resource.
Rights	Information about rights held in and over the resource.
Source	A related resource from which the described resource is derived.
Subject	The topic of the resource.
Title	A name given to the resource.
Type	The nature or genre of the resource.

The *harvest* module stores archive data of all the videos into a redis repository via OAI-PMH. The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [31]. It is a low-barrier mechanism for repository interoperability. Data Provider, as in noozy.tv archive is the repository that exposes structured metadata via OAI-PMH. Noozy AI then make OAI-PMH service requests to harvest that metadata. In our system, harvesting is a scheduled process.

Any algorithm integrated with Noozy AI, will get access to all necessary values from the metadata to compute different types of recommendations in different processes. To note, all the non-ascii characters in it gets encoded into 'utf-8' format. While encoding the resource data into Redis, the metadata fields get 'dc:' as superscript to note that it is in Dublin Core standard. A sample video metadata is given in the appendix chapter.

3.3. xAPI

xAPI is an learning specification that makes it possible to collect data about the wide range of experiences a user has within the portal. The user's likes and dislikes are noted and recorded based on his/her actions like clicks, searches, and views.

The use of an xAPI [32] is required in this case of study to solve several problems faced in the data collection task. Experience data are collected and distributed to the recommenders by following states:

- The platform sends xAPI statements periodically to be stored in the LRS.
- The platform executes predefined methods to create xAPI statements and send them to the LRS.
- The platform uses the implemented endpoints used to create, store, and manage xAPI statements.

The platform communicates with the LRS to gather data about interactions and usage, then sends the collected data to the recommender to apply recommendations. We can also define a bunch of methods that aim to form the corresponding xAPI statements from the provided information, such as played, paused, terminated, connected, searched, liked, rated etc. Here, we have used Trax LRS for this benefit.

3.3.2.1. Trax

Trax LRS is a free and open-source Learning Record Store that focuses on storing and managing data conforming to the xAPI standard. Trax LRS is developer-friendly service and relies on a well-known technology stack. It also permits working with Relational Databases, NoSQL Databases, or Hybrid Databases. Trax [33] is a progressive LRS where data providers can add features, packages, functionalities, etc. Trax LRS collects and tracks the learning experiences complying the standard xAPI specification. Besides, it focuses on storing and managing data basing on the latest xAPI specification.

3.3.2.2. Sample *xAPI Statement*

Statement or learning record the data consists a json like description of 'actor' as in user, 'object' as in video and 'verb' which is the method of the request.

```
{
'actor':
    {
     'mbox':'mailto:user#967553@smartvideo.fr', 'objectType':
      'Agent'},
  'authority': {'account': {'homePage': 'http://trax.test',
                            'name': 'traxlrs'},
                'objectType': 'Agent'},
  'context': {'extensions':
          {'https://smartvideo.fr/xapi/extensions/position':
             'PT0H0M18S'}},
  'id': '7137c119-2c30-301b-8545-b92652b65213',
  'object': {'id':
       'https://smartvideo.fr/xapi/objects/video#5kNs1FpXhq',
             'objectType': 'Activity'},
  'stored': '2021-04-01T06:51:57.8388Z',
  'timestamp': '2021-03-31T14:20:03+02:00',
  'verb': {'display': {'en-US': 'terminated'},
           'id':
              'https://smartvideo.fr/xapi/verbs/terminated'},
  'version': '1.0.0'
}
```

Appendix 3-A. Sample xAPI statement

3.4. Algorithm Module

The algorithm module is the micro-service which collects video metadata and xAPI statements and then computes the predicted video recommendations, in batch with a given schedule and stores it into the result repository. It collects, cleans and process data through integrated algorithm. In all the internal process, all the data are analyzed and computed in Pandas [34] DataFrame data structure.

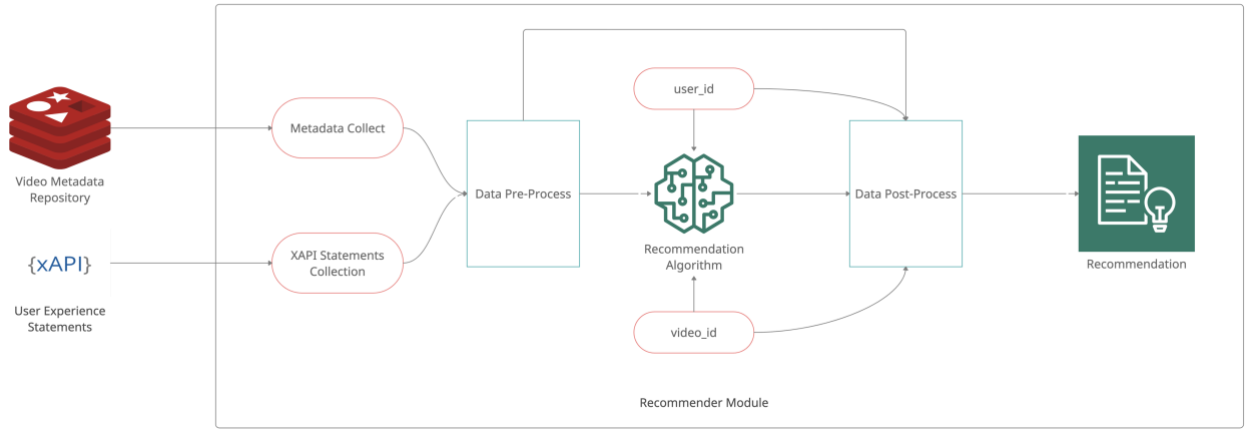

Figure 3.4. Simplified flow of the algorithm module micro-service

3.4.1. Data fetch

In the initialization point of the recommender algorithm process, the module gets all metadata and xAPI statements. It is done once per session, so it dismisses a great deal of the communication overload. Here, only the necessary metadata values for an algorithm are picked for computation. In some basic recommendation, algorithm may need only one or two fields. In advanced, they take more information as input. In that case, the necessary metadata fields of the metadata must be declared in algorithm module configuration. It is a helpful feature of *RedisJSON* and its Python client *rejson*, that we can fetch only the necessary values for all videos from the json data structured.

And for xAPI statements, the Trax server's restful API gives filtered user experience data over the platform's functionalities and traces.

3.4.2. Data pre-process

Video metadata and all the xAPI statements are cleaned and pre-process as per algorithm's need. By removing unwanted data to ease out further processes, pre-process makes the metadata ready for better prediction from algorithms. Merging metadata with xAPI statements can give view matrix, which is the key to filter results per user. View matrix is the matrix which records the user's usage over videos. It defines seen or unseen - these binary labels. For advance algorithms, some other type of matrices could be generated in pre-process part. All the other necessary language processing and any other necessary algorithms to

generate information is also initiated here for a session. In a very basic sense, cleaning the xAPI statements starts by getting user-id and seen video-id.

Table 3 shows a sample metadata of a session, after pre-process. Table 4 and 5 show the figures of view matrix and similarity matrix computed over video's genre for content-based recommendation.

Table 3.2. Sample Metadata after pre-process

Index	Video ID	---	Timestamp
0	s2XESXoDoA	---	2021-05-19T00:00:00+02:00
1	npT2go5wOT	---	2021-02-11T00:00:00+01:00
2	g8t88Ndrcu	---	2021-04-05T00:00:00+02:00
3	R5KcAl8ZU8	---	2021-02-22T00:00:00+01:00
4	GhTnVUoRuL	---	2021-03-20T00:00:00+01:00
		...	
1239	b5XwMhMbZV	---	2021-06-11T00:00:00+02:00
1240	vLrkwj6fpO	---	2021-06-18T00:00:00+02:00
1241	slsG3M8sFh	---	2021-01-18T00:00:00+01:00
1242	sPKXVmpRVy	---	2021-06-04T00:00:00+02:00
1243	JRh0bL2imm	---	2021-03-28T00:00:00+01:00
Length: 1244, dtype: object			

Table 3.3. View Matrix

video-id user-id	0J12RWSAwK	0Pc0pEDehV	zpmIVq9zpD	JRh0bL2imm	GhTnVUoRuL	…	zx7dJiaK8C
10306994	0	1	0	0	0	...	1
9651384	0	1	0	1	1	...	0
..	..	..	..	..	..	...	…
9492955	0	0	0	1	0	...	0
9492917	0	0	0	0	1	...	0

Table 3.4. Genre Similarity Matrix

video-id video-id	0J12RWSAwK	0Pc0pEDehV	zpmIVq9zpD	JRh0bL2imm	GhTnVUoRuL	…	zx7dJiaK8C
0J12RWSAwK	0	1	0	0	0	...	1
0Pc0pEDehV	0	1	0	0	1	...	0
..	..	..	..	..	..	...	…
xNmIkq9kps	0	0	0	1	0	...	0
zx7dJiaK8C	0	0	0	0	1	...	0

3.4.3. Recommender Algorithm Interface

In every different algorithm modules or micro-services, the main distinct part is this python module. It will take a dictionary of strings, integers, Boolean and the DataFrame data structure of required values such as clean metadata, view-matrix, clean xAPI statements, similarity matrix etc. All these data will be used to compute recommendation result as DataFrame object. A recommender algorithm should be placed as python package with a common method to interact with the framework. Which makes every algorithm's process flow same in manner, and data passing through same pattern of objects in the framework. Algorithm module computes all the necessary data passed as key-value paired dictionary data structure to keep dynamic computation easier and low in cost. It will rank the recommendation result with a score depending on algorithm type. To note that, the name of the package will be the algorithm-id. A snippet of the template is as followed

algorithm_id.py

```
import .. ..

def recommend(metadata = None , xapi_statments = None, .. ):
    """
    :param df_videos, xapi_....., matrix.. : DataFrame
    :return: random video list: DataFrame ['hash_key', .. ]
    """
    # result = compute recommendation
    # return result

def get_recommendation(args):
    """
   : return: recommendation results
    """
    return recommend(metadata = args["df_videos"],
                         xapi = args["xapi_statements"],
                         args[".."], .. )
```

Appendix 3-B. Noozy Recommender Algorithm Interface

3.4.4. Recommender Algorithm Blueprint

To make the framework more dynamic for different algorithms and to reduce redundancy in data and communication cost, algorithm module can be configured from a yaml. YAML [35] is an extensively used markup language for configuration files and blueprints. The input and characteristics of each algorithm module is declared here.

Its functionalities are

i. integrating recommender algorithm package with module
ii. naming up algorithm-id
iii. input necessary metadata parameters to compute recommendations
iv. declaring N (required number of result)
v. selecting if User-Based or Item-Based

recommender_algorithm_input.yml

```
input:
  description:
    - "algorithm_id: required"
    - "n: required"
    - "user_id_requirement: 'optional'(default)"
    - "video_id_requirement: 'required' / 'optional' / 'not required'
(default)"
    - "type: 'Basic', 'GenreBased' and 'KeywordBased' = 'ThematicBased' "
    - "dependant_data_parameters - a list of needed parameters for
analysis (required)"
  # name of the algorithm module (.src/algorithm/random_basic.py)
  algorithm_id : "random_basic"
  algorithm_type: "Basic"
  n: 15

  # user_id input - "optional" default

  user_id_requirement: "optional"
  # video_id input - only required for content based / item based
algorithm
  video_id_requirement: "required"
  # dependant_data_parameters (eg. 'dc:description' OR 'dc:available' )
# 'dc:identifier' - default
  dependant_data_parameters : ['dc:identifier',
'dc:title','dc:description', 'dc:type', 'dc:subject', 'dc:available']
  date_range_dependency: "False"
  date_range_before: 30
  date_range_after: 30
  start_from :
    day: 1
    month: 1
    year: 2021
```

Appendix 3-C. Algorithm module blueprint

Making the whole module very compact with parameterized dictation of process is the key functionality of this framework. It can be easily maneuverable and will be very helpful for developers working on this framework.

3.4.5. Data post-process

Data post processing is the step where computation results get easier to understand. The resulting recommended video list gets shaped and encoded as agreed formats, result data gets filtered by user's history and required N.

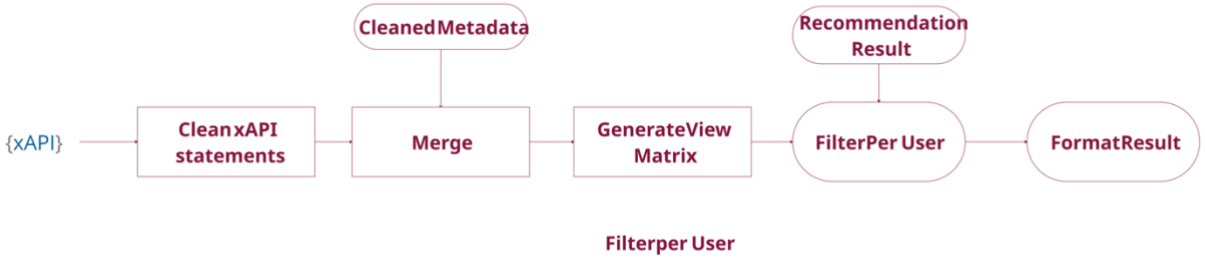

Figure 3.5 Process of filtering and getting unseen videos for each user

For evaluation and research on diversity and new hybrid algorithm, post-process could be the entry point for adding or thinking of new algorithms.

Its functionalities can be

i. filter per user
ii. getting topN
iii. adding extra metadata information of result data
iv. preparing for sending into repository

3.4.6. Result storing

For a heavily data related computation system, data storing and fetching take a good cost of propagation time. Thus, Redis is taken as database for both metadata and result repository. With faster and json structure with RedisJSON, storing in an easily understandable format became easier.

The algorithm module computes recommendation for each user (singed-in) or unauthorized users. So, the task gets bigger and bigger with high number of users. After each user's recommendation generation, the module stores that asynchronously into the repository

having that user's id as KEY. Also, after one session, it generates a log file in json format consisting the information and result of the session.

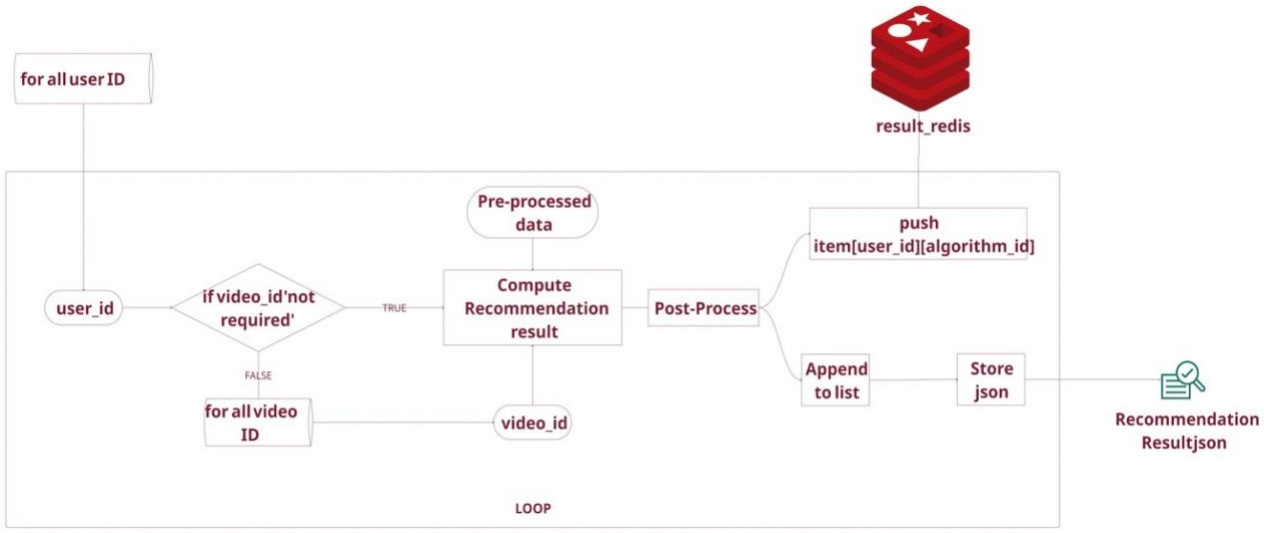

Figure 3.6 Result repository and log saving protocol

3.4.6.1. Log

The session log keeps all the recommended videos by that algorithm module, and other attributes to monitor. Which is used to keep track of the flow of the module's computation.

```
{
  "recommendation_results": [
    {
      "user_id": "None",
      "random-basic": [
        {
          "header": {
            "algorithm_id": "random-basic",
            "N": 5,
            "user_id": null,
            "video_id": null,
            "timestamp": 1622666493.687286
          },
          "result": [
                  { "video_id": "Ug6JPlkODX", "rank": 1, "score": 1.0 },
                  { "video_id": "…", "rank": 2, "score": 0.89 },
                  { … },
                  { … }, ]}]
    },
    {
      "user_id": {$user-id},
      "random-basic": [
        {
```

```
            "header": { … },
            "result": [{ … }, { … }, { … }] } ]
      },
   ],
   "process_start_timestamp": 1622666493.669746,
   "process_finish_timestamp": 1622666493.9715059
   "date": .. ..
   "‟.. ..”: .. ..
}
```

Appendix 3-D. Structure of recommender json result log of one session

3.4.6.2. Result repository

All the recommender algorithm module computes prediction and repository into a redis database. Redis is highly suitable for this type of application. With a simple but json serialized structure, add, update, search, delete etc. fundamental functionalities for serving shows efficiently good performance.

Table 3.5. Structure of the redis repository

Key	Json Encoded Value
User ID	Key: Algorithm ID Value: Recommendations

3.4.7. Module configuration

The credentials for connecting, authenticating and maintain security for communicating with other micro-services or resources, are declared in one *INI* file. An *INI* [36] file is a configuration file for computer software that consists of a text-based content with a structure and syntax comprising key–value pairs for properties. In this development, it gives easy access to integration and structure.

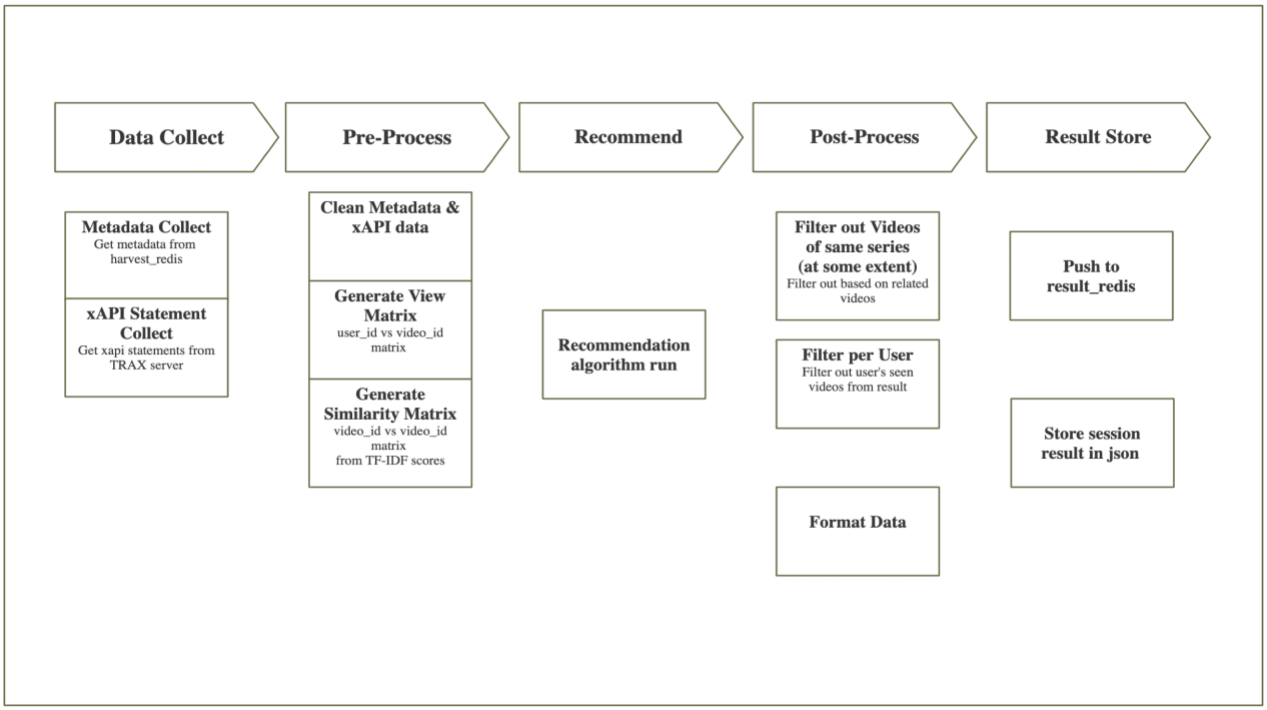

Figure 3.6. Basic process flow of the algorithm module micro-service

To sum up, in one session the, algorithm module's process ends with storing all the recommendation of one algorithm. And, at the next given schedule it does the computation again with the new knowledge and data.

3.5. Scheduler

Scheduler is the script to generate recommendation result, and update repository in a scheduled manner. Different algorithm may need different scheduling depending on algorithm types, viewer's response, platforms requirements etc. For example, recommendation of most recent videos should be frequently computed and updated than the content-based or any hybrid recommendation. It is python script started with a shell script to initiate schedule when build and running of the docker container.

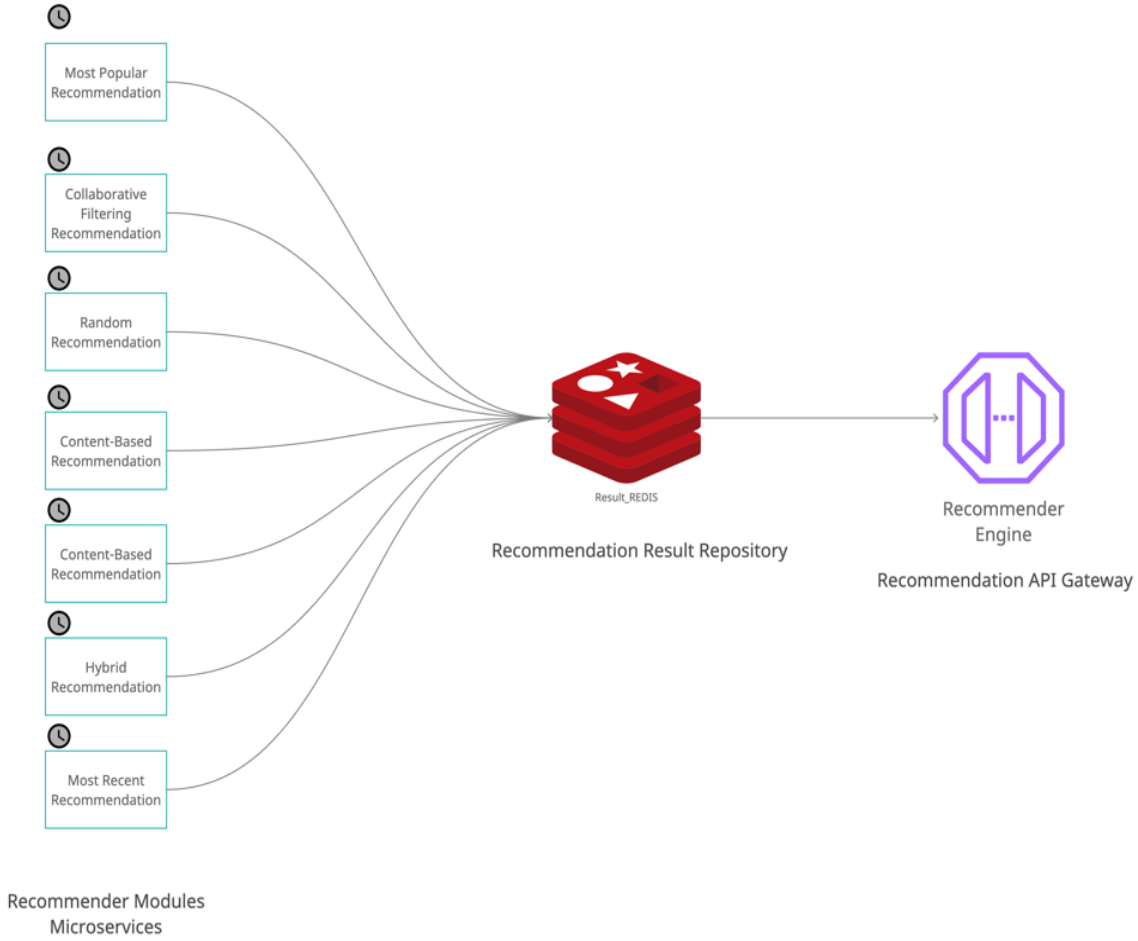

Figure 3.7. Scheduled recommendation compute and repository

3.6. Recommender Engine

Recommender engine is the API gateway to serve recommendation requests. It collects the desired item for recommendation repository and responds the media portal. For programming the server, python flask server is used with openAPI standard RESTful service. Our team used Swagger framework for API design on the flask server.

This application gets recommendation request from the portal. Then it searches the recommendation repository using the inputs from the request. Those inputs contain user-id and algorithm-id and any other necessary values as video-id etc. Then engine searches recommendation result by `item['user-id']['algorithm-id']` from the result-redis.

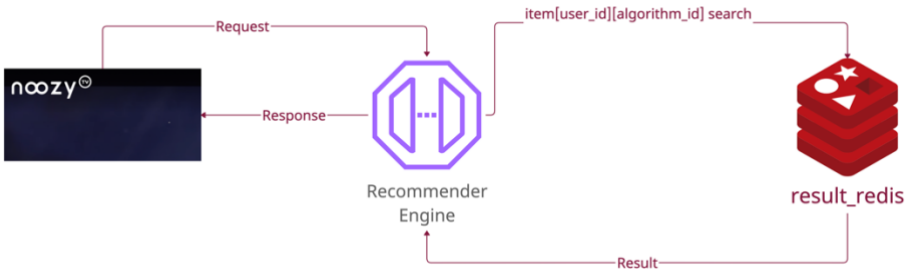

Figure 3.7. Request-response connection between VoD and Recommender

3.6.1. openAPI

The OpenAPI Specification (OAS) [37] is a defined a standard, language-independant interface to RESTful APIs [38] which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.

3.6.2. Swagger server

Swagger server [39] is an open-source and easy API deployment framework. It generates openAPI standard data server to transmit recommendation to the portal. It gives the flexibility to add extra type of recommendation and integrate with request types.

3.7. Integration: Docker and Makefile

Having many micro-services has the issue of installing all of them and maintain updates of them. For easy integration, installation, and update, git comes in handy. All the projects get cloned from private repository in each installation and update. A specific branch in the

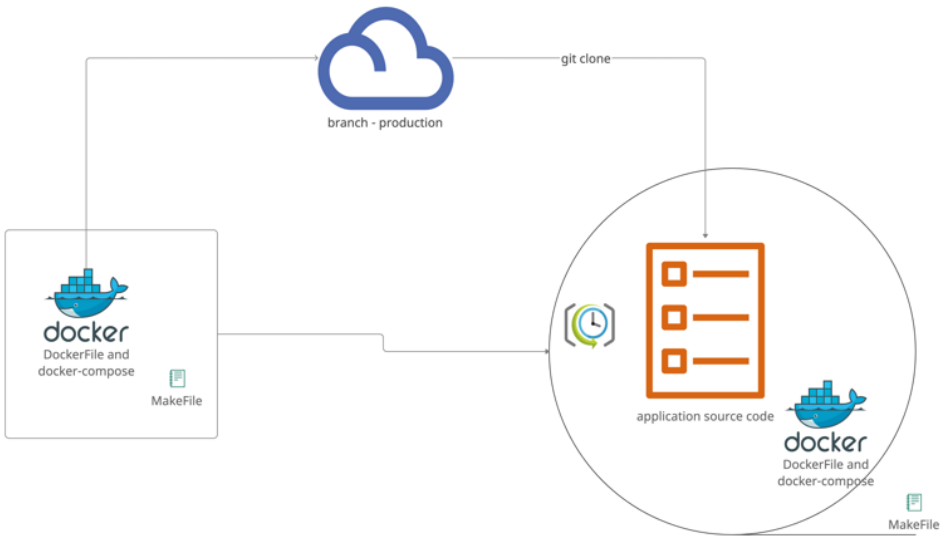

Figure 3.8. Installing micro-service

project's repository gives the permission and install them in server. A *DockerFile* instructs the installation process of the micro-service and a `docker-compose.yml` file used the *DockerFile* to *build* the container. A *Makefile* [40] is used to run and automate the process, along with giving the ability to `make` custom commands integrating the process itself. the docker commands easier. Which makes installation and update, testing and fixing process easier and hassle-free.

4. Reliability

To ensure a sustainable development environment and team, software engineers follow some basic rules for clean and standardized practices while development.

4.1. Code styling standard

For programming the application in Python programming language, PEP8 – Style guide for python [41] is taken for project's standard. A clean and standard coding style plays good role in software engineering team and keep the development sustainable.

4.2. Coding principles

To help development easier, easier to understand and portray expansion of codes, some good practices are considered as principles for coding. It is called SOLID principles [42]. The letters stand for the principles –

i. Single Responsibility Principle:
A class should have one, and only one, reason to change.

ii. Open/Closed Principle
Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

iii. Liskov Substitution Principle
If we have a base class T and subclass S, you should be able to substitute the main class T with the subclass S without breaking the code. The interface of a subclass should be the same as the interface of the base class, and the subclass should behave in the same way as the base class.

iv. Interface Segregation Principle
Clients should not be forced to depend upon interfaces that they do not use.

v. Dependency Inversion Principle
High-level modules should not depend on low-level modules. Both should depend on abstractions (e.g. interfaces).

These are concepts that are easy to understand but very valuable for a good experience in programming software systems. These principles keep the project modular, upgrading guidelines method designing and avoiding boilerplates in an application.

4.3. Modular and Object-oriented programming

For this type of intelligence system, where there are many processes running parallelly and independently, both modular and object-oriented programming is used for the coding. Python being a programming language with functional programming capabilities, coding with both modules and objects makes the framework more subtle and dynamic.

4.4. Tools and libraries

All the necessary libraries and tools are open-source and extensively used in building software solutions like this. Except for Trax server, for the programming of the development – Python and flask[43], rejson, scikit-learn[44], numpy[36], pandas, nltk[45], json, pyyaml [38], configpurser, surprise, swagger-io etc. libraries are used. The environment is created with IntelliJ PyCharm. All the testing and debugging are done in virtual environment of the project

maintaining docker container's specification. For documentation and docstring conversion, Sphinx module is used with IDE. The research and analysis for algorithms are coded and illustrated in Jupyter Notebook. All the external test tools can be operated and used for monitoring from any distant server, as localhost. Trax in built using Laravel framework of PHP.

4.5. Testing

In computer programming, unit testing is a software testing method by which individual units of source code—sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures—are tested to determine whether they are fit for use. For unit testing, the IDE IntelliJ PyCharm's Test Runner module is used. Apart from that, as all the modules are independent, testing them is much easier than monolithic applications.

All the processes store logs of each session. Result log is considered for testing acceptability of the recommendation results. Having *Makefile* in the container, the state of the daemon, logs, errors, security, and bugs can be checked by make commands. Redis CLI is used to monitor and operate the repositories and their functionalities. At the end, with *Postman*, the acceptability of the response for recommendation request is monitored.

This project is designed keeping mind of the fact of a growing team of engineers working on development and testing over this prototype. For starter, checking API response in Postman and evaluating the result by the web tv platform and viewer's standard can be considered as black box testing. And unit testing, monitoring Redis repositories and analyzing the logs the session can be considered as white box testing. But of-course, more time, upgrades, and development on this will bring more test cases, facilities, and roles to play for keeping this application sustainable.

4.6 Backup

In a software system where the independent parallel daemon runs depending on repository data to create new data and repository them, it is crucial to have a systematic backup process. With some simple commands, or in a schedule, all the knowledge and data repositories (metadata, LRS and recommendations) can be stored in a backup repository. Which will prevent data loss in case of any kinds of error.

5. Case Study

For development, testing and presenting the algorithm modules and all the other mico-services working independently together, Content-Based video recommendation is taken as use case recommender algorithm. The python project file of the algorithm follows the template interface and integration with the blueprint YAML file.

5.1. Content-based Recommendation

The characteristic of this algorithm is to deliver a video list sorted by the similarity of contents, as in information from metadata with one or some given reference video. The reference video can be any video in the platform naming ItemBased and could be N number of user's seen videos over a period, which is UserBased. Based on the fields of which value's similarity would be used in the computation, we got recommendation by genre, keyword, and of-course the description or contents of the video in metadata. So, we get, some variations of content-based recommender family which are given specific algorithm-id such as, `CB-ItemBased GenreBased`, `CB-ItemBased-KeywordBased`, `CB-UserBased-Basic` etc.

To compute genre based or keyword-based filtering, we generate similarity matrix. Which is a video-id vs video-id matrix based on binary or integer labels, which denotes one video is in the other videos same genre or have similar keywords. As keyword or genre are list of small strings in the metadata [appendix10-A], it is easy to get similarity matrix. But for the basic content-based algorithm which takes video's description into account the process is bigger.

For this algorithm, we only need video-id, description, genre, and keyword values from the metadata.

5.2. TF-IDF score

The TF-IDF [46] algorithm is used to weigh a keyword in any document and assign the importance to that keyword based on the number of times it appears in the document. Put simply, the higher the TF-IDF score (weight), the rarer and more important the term, and vice versa. Each word or term has its respective TF and IDF score. The product of the TF and IDF scores of a term is called the TF-IDF weight of that term. The TF (term frequency) of a word is the number of times it appears in a document. When you know it, you're able to see if you're

using a term too often or too infrequently. The IDF (inverse document frequency) of a word is the measure of how significant that term is in the whole corpus.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).
IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

$$W_i = tf_i \times (ln\frac{N+1}{df_i} + 1)$$

tf_i = frequency of x in y

df_i = number of documents containing x

N = total number of documents

In Python, scikit-learn provides you a pre-built TF-IDF vectorizer that calculates the TF-IDF score for each item's description, word-by-word. The feature vector represents TF-IDF vector of the document "natural language processing action understanding analyzing generating text python". The benefit of converting this document into a vector is that we can now use dot product to calculate the cosine similarity. Moreover, representing a document in vector format opens up the possibility to use many other mathematical models which operate on numeric data. There are many words in a document that occur many times but may not be important; in English, these are probably words like "the", "is", "of", and so forth. We might take the approach of adding words like these to a list of stop words and removing them before analysis, but it is possible that some of these words might be more important in some documents than others. A list of stop words is given for a sophisticated approach to adjusting term frequency for commonly used words. Considering the portal and int's contents language is French, we have used *nltk* corpus [45] library' recorded stop words for French document vectorizing. Table 5.1 is a sample pre-processed tf-idf vector.

Table 5.1. Sample Tf-IDF scores of terms in description

	term	score
0	ã©vã¨nements	0.263941
1	bonaparte	0.249586
2	beaux-arts	0.249586
3	napolã©on	0.231501
4	mãªme	0.148592
...	...	...
11487	eco-citoyen	0.000000
11488	eco-rã©gions	0.000000
11489	ecologie	0.000000
11490	ecovelo	0.000000
11491	å“uvres	0.000000

Table 5.2. Sample Tf-IDF scores of video description by Index

Index	Tf-IDF score
(0, 5526)	0.1563741483444983
(0, 290)	0.16110938411386022
(0, 5544)	0.1895736174216862
(0, 4769)	0.24598407552065216
(0, 3980)	0.2230476955572924
(0, 8425)	0.1444615852883775
(0, 8058)	0.1485519960155001
(0, 9314)	0.11491996912353665
:	:
(1243, 9635)	0.10331498447096109
(1243, 8204)	0.10008085670918328

5.3 Cosine Similarity Matrix

Cosine similarity [47] is a vector-based measure of the similarity of two strings. The basic idea behind cosine similarity is to transform each string into a vector in some high dimensional space such that similar strings are close to each other. The cosine of the angle between two vectorsis a measure of how "similar" they are, which in turn, is a measure of the similarity of these strings. If the vectors are of unit length, the cosine of the angle between them is simply the dot product of the vectors. There are many ways of transforming a string in the database into a vector. The tf.idf vector is a popular choice for this representation. The tf.idf vector is composed of the product of a term frequency and the inverse document frequency for each token that appears in the string.

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \,.\, \vec{y}}{|\vec{x}| \,|\, |\vec{y}|} = \frac{\sum_{i=0}^{N} x_i \times y_i}{\sqrt{\sum_{i=0}^{N} x_i^2}\ \sqrt{\sum_{i=0}^{N} y_i^2}}$$

x and y = tf-*idf* vectors of item i

It computes the dot product of this vector with the vector for each row in the matrix – this is the cosine similarity. For this, scikit-learn's PairwiseCosineSimilarity library is used. Besides working with only cosine similarity methods, research, and development of Hybrid Semantic text similarity method with collaborative filtering is also taking place on the system. Table 5.1 shows the index vs index matrix of cosine similarity scores. These are considered to be the scores of the recommended videos.

Table 5.3. Cosine Similarity Matrix of Tf-IDF scores of Description (Index vs Index)

index index	0	1	2	3	4	…	1242	1243
0	1.000000	0.017429	0.005979	0.004390	0.024523	...	0.013097	0.000000
1	0.017429	1.000000	0.002653	0.005017	0.012032	...	0.000000	0.005568
.	..	..	..	..	..	…	..	..
.	..	..	..	..	..	…	..	..
1241	0.005979	0.002653	0.003000	0.007845	0.002013	...	0.000000	0.000000
1242	0.004390	0.005017	0.007845	0.000679	0.008413	...	1.000000	0.016756
1243	0.024523	0.012032	0.002013	0.008413	0.007600	...	0.025548	0.016091

5.4. Algorithm

Firstly. it takes all the clean metadata, and pre-processed data into consideration. According to the type of algorithm selected from the content-based recommender family, it will use the matrices to compute recommendation. It will give ItemBased recommendation for each video item and UserBased recommendation for reference videos of each user. For evaluation purpose, different field values are computed for study. Figure 5.1 describe the flow of this algorithm.

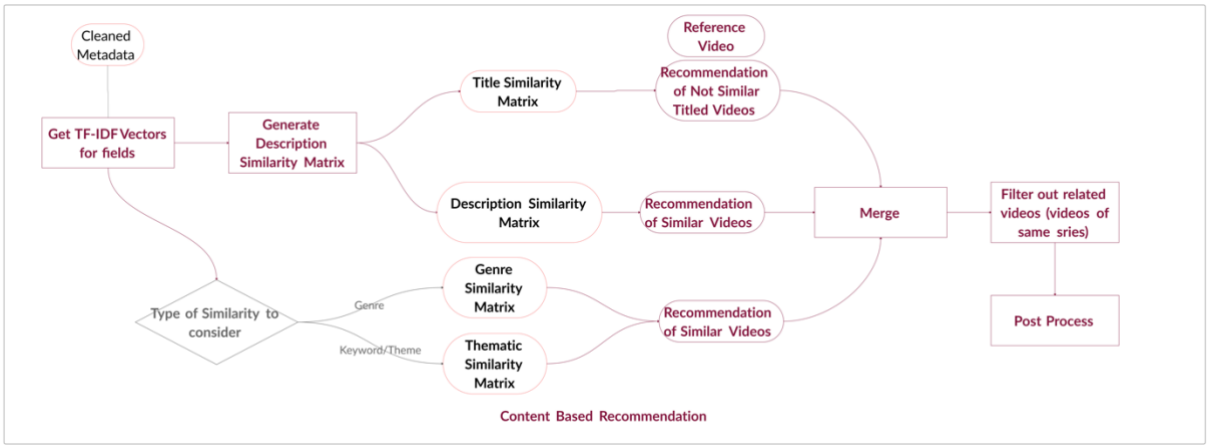

Figure 5.1. Flow diagram of content-based algorithm

The recommender process is illustrated simply, using this pseudocode (Appendix 5-A).

```
video_id[x] = reference video id / list of ids

similarity matrix:
     generate tf-idf vectors for each video's description
     return cosine similarity matrix 'tf-idf[x] X tf-idf[y]'



recommender (matrix, reference-video-id):
     for all video-id in matrix[reference-video-id]:
          sort by similarity score
     result = {'video_id', 'similarity score'}
```

```
return result

description-based-recommendation:
     return recommender (description similarity matrix, ..)

genre-based- recommendation:
     return recommender (genre similarity matrix, ..)

keyword-based-recommendation:
     return recommender (keyword similarity matrix, ..)
```

Appendix 5-A. Pseudocode of used Content-Based Recommender

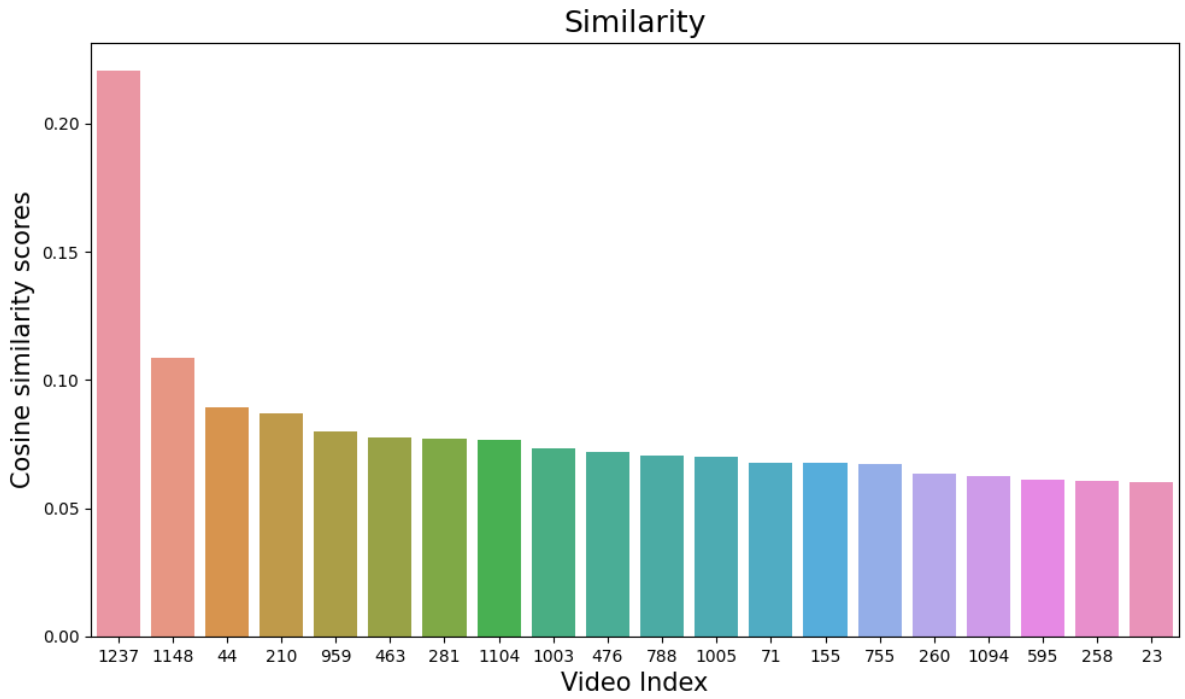

Figure 5.2 Bar chart showing similarity of video description to a reference video

This bar chart (Figure 5.2) shows the video list by sorted similarity scores to a reference video. For `ItemBased` type in this case, we take only one reference video. But, in case of `UserBased`, we can take user's N number of last seen videos or favorite videos etc. and merger their results.

5.5. Post-process

Before the process of filtering and structuring the results, the module joins one result with another to compute and sort for different basic hybrid results if denoted in the blueprint. Which gives sparse values with various algorithms to get evaluation on.

```
genre-description-recommendation:
     return sort (merge (description-based-recommendation,
                         genre-based- recommendation))

keyword-description-recommendation:
     return sort (merge (description-based-recommendation,
                    keyword-based-recommendation))
```

Appendix 5-A. Pseudocode of used extra post-process for some basic-hybrid algorithm

6. Evaluation

6.1 Online Evaluation

In online evaluation, the system measures the change in user behaviour when interacting with different recommender systems. In the case of rating prediction tasks, the value of such predictions can depend on a variety of factors such as the user's intent (e.g., information needs, novelty vs risk), the user's context (e.g., familiar items, system trust), and the interface through which the predictions are presented. The advantages of online experiment are that the entire performance of the recommender can be evaluated such as long-term business profit and users' retention. Therefore, online experiment can be used to understand the impact of evaluation metrics on the overall performance of the system. The progressive evaluation process will reduce the risk of online experiment and accomplish satisfying recommendation results.

However, in a multitude of cases, such experiments are very costly, since creating online testing systems may require much effort. Furthermore, we would like to evaluate our algorithms before presenting their results to the users. For example, a test system that provides irrelevant recommendations, may discourage the test users from using the real system ever again. For these, we will present online evaluation procedures to Noozy TV's designated personnel. We set up some sets of recommendation with variety of basic and hybrid algorithms on content-based recommender family. Each algorithm will have 15 reference video and their top 15 recommendations. The only the development team knows the mappings of algorithm-id and those sets of results. Moderator can rate each resulting video for its rank and can comment on each result sets.

After getting the evaluation scores, we can decide to deploy one or make updates to one to increase accuracy up to the requirements.

6.2. Offline Evaluation

The goal of the offline evaluation is to filter algorithms so that only the most promising need undergo expensive online tests. Thus, the data used for the offline evaluation should match as closely as possible the deployed online. Ensuring that there is no bias in the distribution of users, items and ratings, the experimenter may be tempted to prefilter the data by excluding non-interesting items. In order to evaluate algorithms offline, it is necessary to simulate the online process were the system makes predictions or recommendations, and the user corrects the predictions or uses the recommendations. This is usually done by recording historical user data, and then hiding some of these interactions to simulate the knowledge of how a user will rate an item, or which recommendations a user will act upon.

Here, the similarity scores of the results can be the evaluation parameter in this certain algorithm, but as a part of cultural media market, we must respect the response of users and authorized personnel of Noozy TV. For Content-Based recommendation, the score of each recommendation can be a factor for evaluation metrices. But also in any case, Noozy TV is still has a very low number of users. On the other hand, AI simulated user data will not be fit for proper evaluation. Thus, the research and development of the platform will continue offline evaluation after a certain period, upgrades and when the necessary situations meet.

7. Hybrid Recommendation

With this framework, research on hybrid algorithms could be hassle-free. Just some input parameter and post-processing steps can be configured to generate hybrid recommendation.

```
merge Most-Popular with Content-Based recommendation

merge Most-Popular with Content-Based recommendation

merge Item Based Collaborative filter with Content-Based
recommendation

merge Nearby Collaborative filter with Content-Based
recommendation
```

Appendix 7-A. Pseudocode of possible hybrid recommender algorithm

With the emergence of lots and lots of information variables in *knowledge* and *data* basic recommender algorithms should be outperform by hybrid algorithms and provide users better experience.

8. Diversity

Diversification has become one of the leading topics of recommender system research not only as a way to solve the over-fitting problem but also an approach to increasing the quality of the user's experience with the recommender system. [47] After developing the Noozy AI, the team is and will research on diversity in recommender system. Concretely, taking the local usage data, xAPI and knowledge, A few promising approaches are in development to try and solve the stability-accuracy-diversity [48] of a recommender system.

Adding and researching on diversity on this framework will be hassle-free. Right after the post-process part of any algorithm module, any developed diversity algorithm can be easily integrated or, it can be another independent module to work with.

9. Conclusion and Future works

The requirement of this project is to provide a dependable recommender system for Noozy TV, a video-on-demand platform for local viewers of Grand EST region of France. That said, our aim is to develop and place a sustainable software engineering environment for research and development over this platform Noozy AI. Having so little data in our hand, it is tough to evaluate some criteria of the experiments unlike other popular web media platforms. But, with time this project could be the base of implementing new algorithms and trends. Over this platform, team BIRD aims to increase the credibility of this platform with

- Dashboard web application for management
- Coming up with a Platform-as-a-Service
- Study on diversity on recommender system
- Remote logging methods and functional commands
- Study on hybrid recommender system and their application

9. References

[1] M. Turner, D. Budgen and P. Brereton, "Turning software into a service," in Computer, vol. 36, no. 10, pp. 38-44, Oct. 2003.

[2] Xie, Tao , “Intelligent Software Engineering: Synergy Between AI and Software Engineering” in Dependable Software Engineering. Theories, Tools, and Applications, pp 3-7. between AI and Software Engineering

[3] noozy.tv

[4] Li-Shen Juhn and Li-Ming Tseng, "Harmonic broadcasting for video-on-demand service," in IEEE Transactions on Broadcasting, vol. 43, no. 3, pp. 268-271, Sept. 1997.

[5] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. Commun. ACM 35, 12 (Dec. 1992), 61–70.

[6] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In Proceedings of the 1994 ACM conference on Computer supported cooperative work (CSCW '94). Association for Computing Machinery, New York, NY, USA, 175–186.

[7] Peis, E., del Castillo, J. M., and Delgado-L pez, J. A. Semantic recommender systems. analysis of the state of the topic, Pp 1–5, 2008

Grouplens: an open architecture for collaborative filtering of netnews. In Proceedings of the 1994 ACM conference on Computer supported cooperative work, pages 175–186.

[8] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2000a). Analysis of recommendation algorithms for e-commerce. In Proceedings of the 2nd ACM conference on Electronic commerce, pages 158–167.

[9] www.kardham-digital.com/

[10] Srisakaokul, S., Wu, Z., Astorga, A., Alebiosu, O., Xie, T.: Multiple-implementation testing of supervised learning software. In: Proc. AAAI-18 Workshop on Engineering Dependable and Secure Machine Learning Systems (EDSMLS), 2018

[11] Pei, K., Cao, Y., Yang, J., Jana, S.: DeepXplore: Automated whitebox testing of deep learning systems. In: Proc. Symposium on Operating Systems Principles (SOSP). pp. 1-18, 2017

[12] Dragoni, Nicola, Giallorenzo, Saverio, Alberto Lluch and Safina, Larisa, "Microservices: Yesterday, Today, and Tomorrow" in Present and Ulterior Software Engineering, Springer International Publishing, pp 195-216, 2017

[13] martinfowler.com/articles/microservices.html

[14] Carl Boettiger. 2015. An introduction to Docker for reproducible research. SIGOPS Oper. Syst. Rev. 49, 1 (January 2015), 71–79.

[15] www.freedesktop.org/software/systemd/man/daemon.html

[16] redis.io

[17] www.json.org/json-en.html

[18] oss.redislabs.com/redisjson/

[19] www.jetbrains.com/pycharm/

[20] jupyter.org

[21] D. Spinellis, "Version control systems," in *IEEE Software*, vol. 22, no. 5, pp. 108-109, Sept.-Oct. 2005

[22] gitlab.inria.fr

[23] H. Liu and H. B. Kuan Tan, "Covering code behavior on input validation in functional testing," Information and Software Technology, vol. 51, no. 2, pp. 546–553, Feb. 2009.]

[24] Nidhra, S., Black Box and White Box Testing Techniques - A Literature Review. International Journal of Embedded Systems and Applications, pp 29-50, 2012

[25] D. Shao, S. Khurshid, and D. E. Perry, "A Case for White-box Testing Using Declarative Specifications Poster Abstract," in Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007, 2007, p. 137.

[26] www.gnu.org/licenses/gpl-3.0.en.html

[27] fsf.org/

[28] www.sphinx-doc.org/en/master

[29] www.w3.org/RDF

[30] www.dublincore.org/resources/glossary/dublin_core

[31] www.openarchives.org/pmh

[32] fraysse.eu/fr/xapi-en-bref

[33] traxproject.fr/

[34] pandas.pydata.org

[35] learn.getgrav.org/17/advanced/yaml

[36] fileinfo.com/extension/ini

[37] www.openapis.org/

[38] restfulapi.net

[39] swagger.io/
[40] www.gnu.org/software/make/manual/make.html
[41] www.python.org/dev/peps/pep-0008
[42] Gary McLean Hall, "Adaptive Code: Agile coding with design patterns and SOLID principles" Microsoft Press, 2017
[43] flask.palletsprojects.com/en/2.0.x/
[44] Pedregosa, F., Varoquaux, G. and Gramfort, et al, "Journal of Machine Learning Research", v-12, pp 2825—2830, 2011
[45] Bird, Steven, Edward Loper and Ewan Klein, "Natural Language Processing with Python" in O'Reilly Media Inc., 2009
[46] Akiko Aizawa, "An information-theoretic perspective of tf–idf measures" in Information Processing & Management, Volume 39, Issue 1, pp 45-65, 2003
[47] Matevž Kunaver, Tomaž Požrl, "Diversity in recommender systems – A survey" in Knowledge-Based Systems, Volume 123, pp 154-162, 2017
[48] Lei Hou, Kecheng Liu, Jianguo Liu, Runtong Zhang, "Solving the stability–accuracy–diversity dilemma of recommender systems" in Physica A: Statistical Mechanics and its Applications, Volume 468, pp 415-424, 2017

10. Appendix

Sample Video Metadata


```
    {
        "dc:title": "C\u00c3\u0083\u00c2\u00b4t\u00c3\u0083\u00c2\u00a9
              sports : Jean-Charles Mourot & Alexandre Klein (22/02)",
        "dc:identifier": "R5KcAl8ZU8",
        "dc:subject": [
            {
                "identifier": "TH0007",
                "title": "Sport"
            },
            {
                "identifier": "TAG000510",
                "title": "biathlon"
            },
            {
                "identifier": "TAG000687",
                "title": "Football"
            },
            {
                "identifier": "TAG001560",
                "title": "GESN"
            },
            {
                "identifier": "TAG000363",
                "title": "SAS Volley"
            }
        ],
        "dc:description": "Avec Jean-Charles Mourot,
pr\u00c3\u0083\u00c2\u00a9sident de l'AS
Plombi\u00c3\u0083\u00c2\u00a8res Football, Alexandre Klein, central du
SAS Volley & … … de l'AS d'ann\u00c3\u0083\u00c2\u00a9e 2021 et le
parcours d'Alex Klein",
        "dc:publisher": [
            {
                "identifier": "ME0000019",
                "title": "VosgesTV"
            }
        ],
        "dc:contributor": [
            {
                "identifier": "ME0000268",
                "title": "VosgesTV"
            }
        ],
        "dc:created": "2021-02-25T10:37:52+01:00",
        "dc:modified": "2021-03-03T15:59:09+01:00",
        "dc:available": "2021-02-22T00:00:00+01:00",
        "dc:type": [
            {
                "identifier": "ME0000628",
                "title": "Magazine/\u00c3\u0083\u00c2\u00a9mission"
            }
        ],
        "dc:extent": "00:27:09",
        "dc:coverage": [
```

```
            {
                "lat": 48.17006,
                "lng": 6.484474
            }
        ],
        "dc:licence": "free",
        "dc:rightsHolder": [
            {
                "identifier": "ME0000026",
                "title": "VosgesTV"
            }
        ],
        "dc:audience": "departement",
        "dc:source.identifier": "EMI000207",
        "dc:source.title":
          "C\u00c3\u0083\u00c2\u00b4t\u00c3\u0083\u00c2\u00a9 sports",
        "dc:issued": NaN,
        "dc:creator": NaN,
        "dc:relation": NaN
    }
```


Appendix 10-A. Sample harvested metadata of a video