Data Analytics Pipeline - First Iteration

Project: Youtube Channel Recommendation System

- Aashna Kanuga (adk2159)
- Akshata Patel (amp2313)
- Kiran Saini (ks3630)
- Neha Saraf (ns3308)

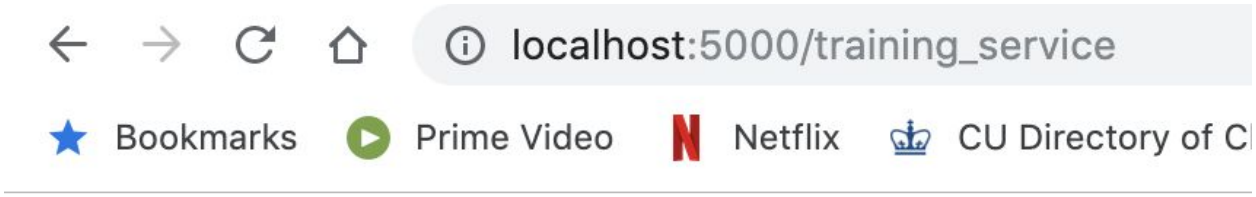
1. **JIRA link:**
   https://toydemoproject.atlassian.net/jira/software/projects/YCR/boards/23

2. **GitHub link:**
   https://github.com/nehasaraf1994/DataPipeLineProject

3. **Steps to run the project:**
   We have the data uploaded in a MySQL database. The server runs on the localhost. So to connect the prediction service with the SQL server we use the details of the database, added in the predict_service.py file to connect to mysql (host="127.0.0.1", user="root", passwd="xxxxx", db = "DataPipeline"):

   A. The following files need to be in the same folder:
      - training_service.py
      - get_product_data.py
      - predict_service.py
      - app.py
      - extract_data.py
      - get_thumbnail_tags.py
      - indico.py
      - A folder “templates” containing the file “home.html”

   B. Run the training service by using the following command on the terminal:
      - python3 training_service.py
      - Go to the browser “http://localhost:5000/training_service”. This will store the trained model in a pickle file on the local folder.
      - The image below is the output of the training service once it has finished running



C. Run the get_product_data.py on the terminal, to set up the GET API, which will be called from the frontend:

- python3 get_product_data.py
- Keep this file running in one terminal window

D. To run app.py, we will use its dockerized version:

a. Run the following command to create an image for a container:
docker build -t *your-docker-image-name* **.**

b. Run the following command to run the container:
docker run -d -p 5000:5000 *your-docker-image-name*

4. **Screenshots**

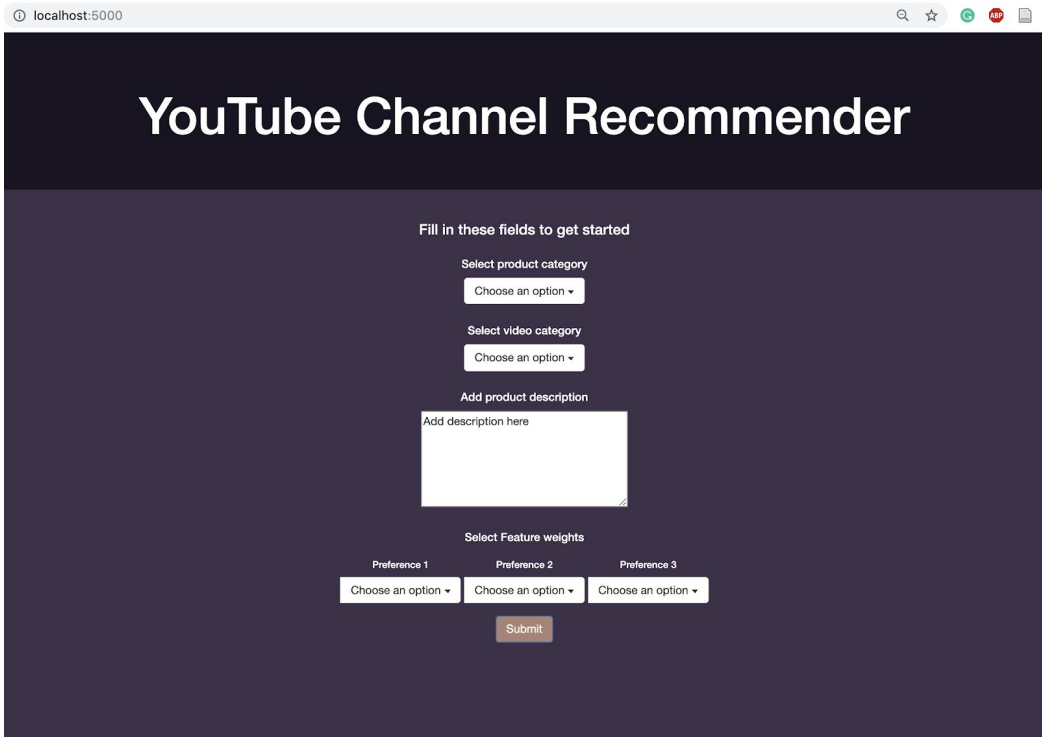


Fig 1. An initial look at the front-end

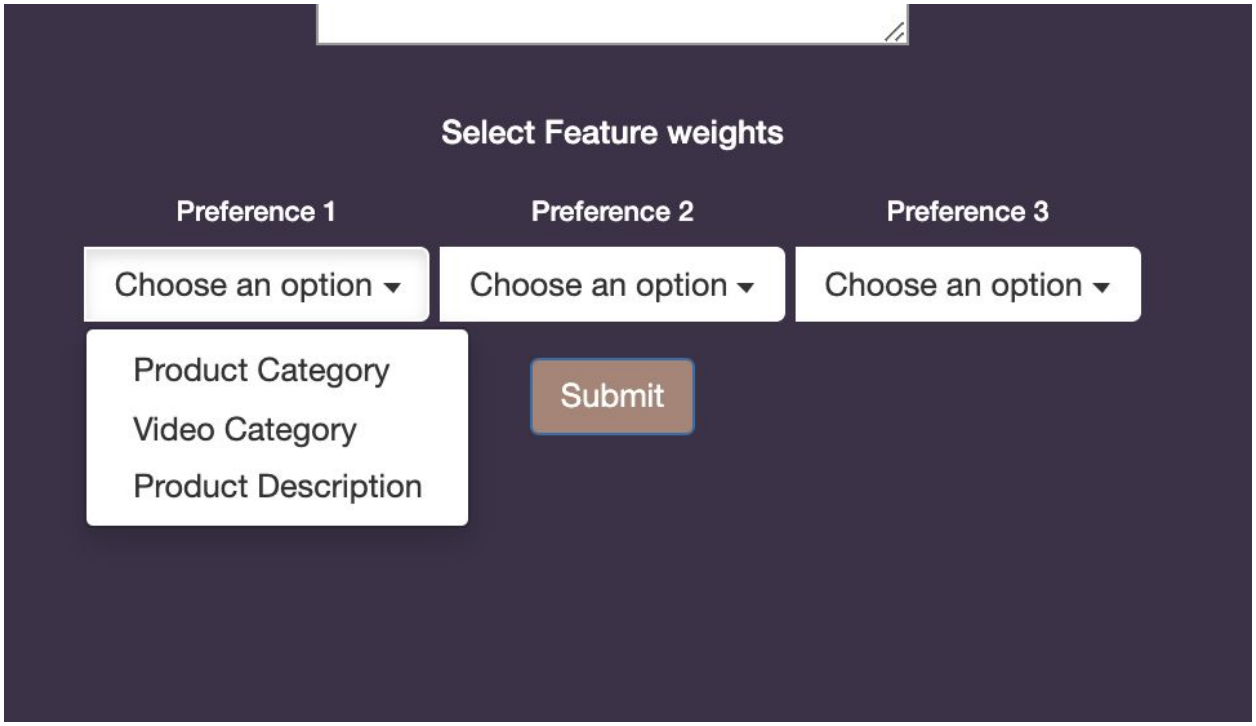


Fig 2. A look at the user preferences feature

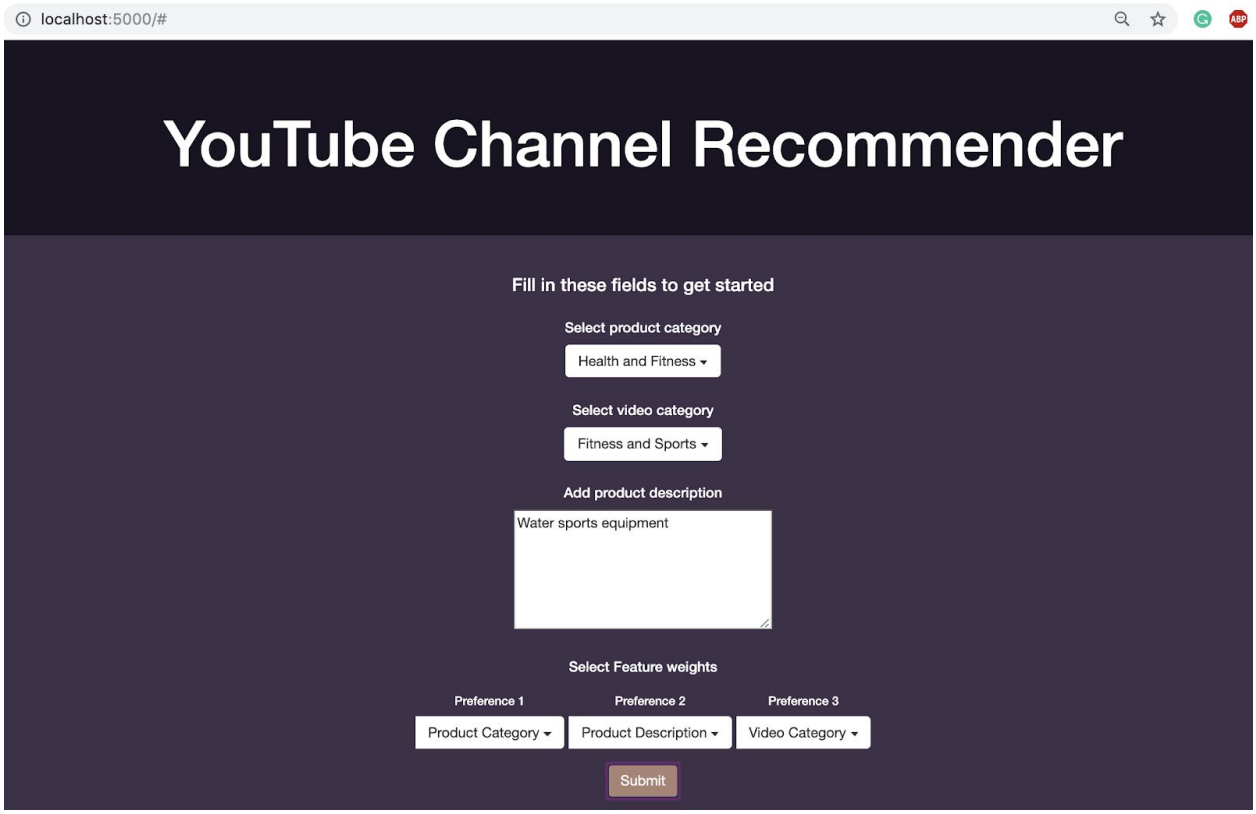


Fig 3. Input was given by the user and submitted

Fig 4. The top 5 channels recommended and the analysis of the channels - Average Like and Dislikes

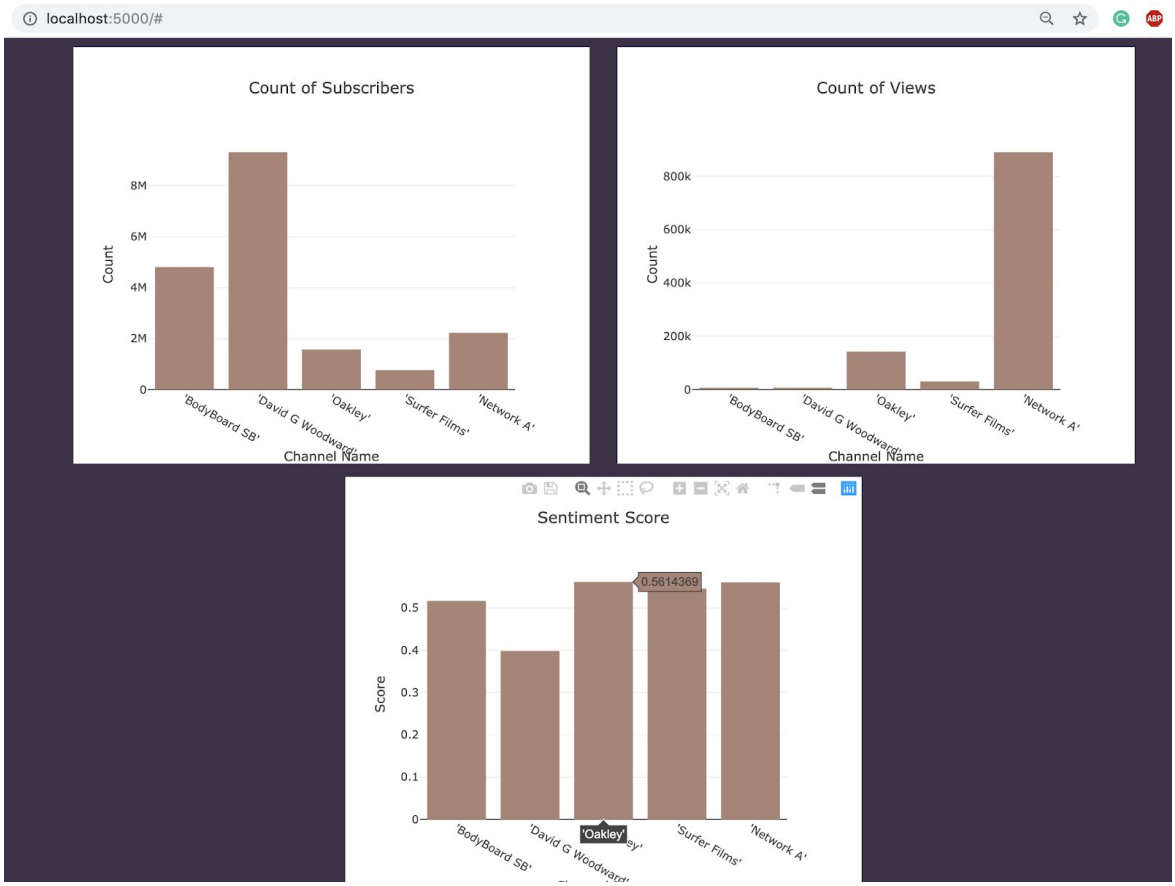


Fig 5. Further analysis, showing the number of subscribers, Average views and average sentiment score. (graphs are interactive)

5. **Description:**

The following new features have been added after the MVP:

- Scraping latest data (to be repeated every six months) from youtube instead of a static data set.
- Extra features such as the thumbnail tags were computed.
- Taking new inputs from the user for the product description and the feature importances. The feature importances are the preferences the user wants to give to the different features : product category, video category and the product description. That is, if the user selects “A” as his first preference, “B” as the second and “C” as the third preference, the code would assign a weight of 0.5 to “A”, 0.3 to “B” and 0.2 to “C”.
- We also calculated sentiment score of each video based on the sentiment of user comments.

- Displayed various statistics of the recommended channels on the dashboard.
- Individual services are dockerised.

Updates to the individual APIs:

1) Dashboard : home.html file
    a) Two new user inputs are added to the home html page for the product description and the feature preferences. The weights are computed based on these preferences. These feature weights are used in the Recommendation Service.
    b) When the user clicks on the “Submit” button, these four features : product category, video category, product description and the features weights are sent as arguments to the recommendation service using the API of the get_product_data python file.
    c) The results(top 5 recommendations) returned from this API are displayed in a table on the dashboard along with the bar charts of the count of likes, dislikes, views, subscribers and sentiment score for each channel recommended.

2) Youtube Data API : extract_data.py
    This program uses the Youtube Data API to extract these features from the latest youtube videos - 'video_id', 'title', 'channel_title', 'category_name', 'tags', 'views', 'subscribers', 'likes', 'dislikes', 'thumbnail_link', 'comments'.
    Note: The free version allowed us to pull only a certain amount of data at a time due to which the dataset is smaller in size than it would be in a real world scenario. Even then the recommendations are fairly accurate.

    We have used different endpoints of Youtube Data API to extract the required information. Endpoints which we have used in the order:
    a. Search : This extracts all the videos posted recently with the region code as United States. With one api call, we extract 50 such results.
    b. Videos: Now we take each video from Search results and extract their information from videos endpoint. This takes the video id as an input parameter to extract these details about the uploaded video: Video Title, Thumbnail Link, Channel Title, Tags, Category ID, Views, Likes, Dislikes, Channel Id.

c. Channels: Now to extract the subscribers for that particular channel, we have used the Channels endpoint - this takes the channel id as the input parameter. (Note: we get channel id from the videos endpoint api)
d. VideoCategories: To extract the video category title, we have used the endpoint VideoCategories which takes in the input parameter as category_id (Note: It will always be a unique value for a particular category id)
e. CommentThreads: This extracts the top 100 comments for a particular video id

3) Pre-computing two features for the data: pre-compute-for-data.py
   The program calls the indico API and Microsoft Cognitive API described below to pre-compute two features of the YouTube dataset:
   a) Get average sentiment of each video
   b) Get tags of the thumbnail of each video

4) Extracting thumbnail Tags: get_thumbnail_tags.py
   The program reads in a url of an image whose tags we want to extract. The Microsoft Cognitive Services Vision API is used to get the tags of the image. The parameters passed are {'VisualFeatures':'Description'}. The output from the API is converted to a json object. A list of tags can be extracted from the description object using key 'tags'. The list of tags is returned as the output.

5) Indico API: indico.py
   This program uses the Indico API for performing text analysis on both the user data and the youtube comments extracted from the Youtube API.
   a) The get_sentiment function reads in a list of comments for a single video, calls the Indico API and gets the sentiment value for each comment in the range 0 to 1. (0 means negative, 1 means positive). All of these values are averaged out to get the mean sentiment score of each video.
   b) The get_keywords function takes in the product description given by the user and extracts the top 5 most important keywords along with their weightage.

Note: The Microsoft Cognitive API and Indico API have limited credit as well. This also prevents us from increasing the size of our database.

6) Prediction and Recommendation Service : predict.py
    a) The get_similarity_score function was updated to include the following functionalities:
        i) Calculating the similarity between the product description and category names by taking a weighted average for each keyword in the description and its similarity with all category names.
        ii) Calculating the similarity between the product description and the video tags in the same way as above, and taking the average of the top five most similar tags.
        iii) For product category, video category as well as product description, we also get the similarity scores with the extracted keywords from the video thumbnail tags.
        iv) Instead of simply average out the similarity scores for all three user inputs, we take a weighted average based on the preferences provided by the user.

    b) The predict function now also calls the indico API to extract the keywords from the product description given by the user.

    c) The recommend_channel function was initially returning just the list of recommended channels to the dashboard. Now it also returns a list of average likes, dislikes, views, subscribers and sentiment scores of the top five most similar channels.

7) Dockerizing all the micro services :

- To dockerize the application we have written two required files: Dockerfile and requirements.txt
- Dockerfile will contain the required files which we want to run. Requirements.txt will contain all the different packages to be required for the application to run.
- Dockerization will make sure the application can be deployed in any of the platforms.
- After writing the the two files:
    b. Run the following command to create an image for a container:
    docker build -t *your-docker-image-name* **.**
    b. Run the following command to run the container:
    docker run -d -p 5000:5000 *your-docker-image-name*
- Few drawbacks of our dockerized application:

Not all the services have been dockerized. This is due to multiple services running simultaneously. This can be done using docker-compose but due to lack of documentation and proper understanding, that could not be done. (Faced a lot of errors). Though the main application is dockerized.