BLIND-FRIENDLY
VIDEO RECOMMENDATION
APPlication

Class of 2024, Natural language processing

Artificial Intelligence & Data Science, Jio Institute

Table of Contents

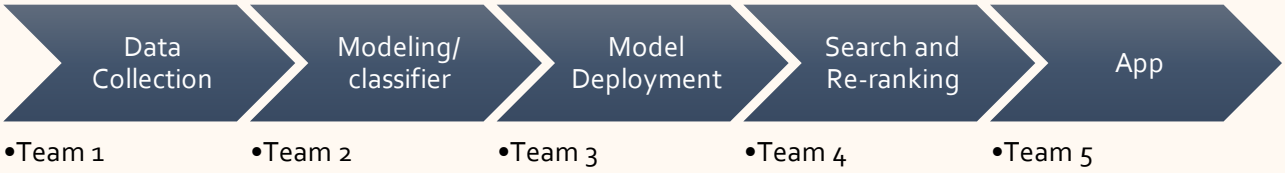
AI & DS 2024

INTRODUCTION

Ever imagined how It feels like to live in the dark? We started with this question to explore how the digital products we use everyday are seen or used by blind people. Here in this project we look at making the Youtube blind friendly. To start with a huge corpus of labelled data was created with features like transcript, genre, title, description, length, and percentage of audio content. Next, DistilBERT model was used to extract embeddings from video transcripts. LIME was used for model explanation. Furthermore, the model was deployed on cloud and API was created that ranks YouTube videos on blind friendly parameters. On top of this, UI was added with audio to text, and text to audio capabilities.

In summary, this project gave us insights into the product design complexities and technology interplay at scale. Future score includes improving the accuracy of the model and adding better UI features by tracking the performance of the model and application usage

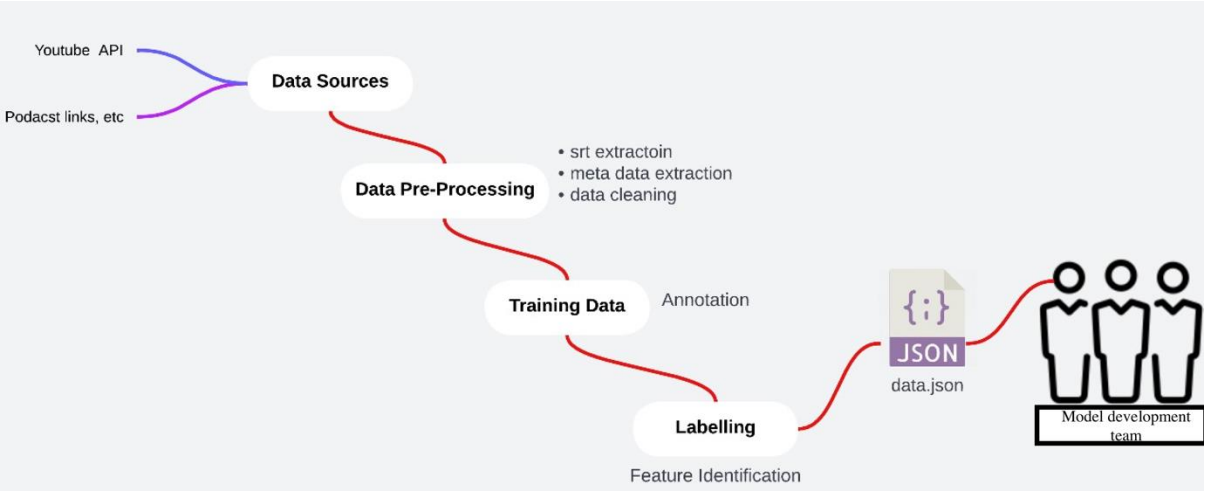
Project Delegation:



Team 01: Data Collection

Objective:

The project aims to develop an application that caters to the visually impaired by recommending blind-friendly videos. Our team's specific responsibility is to create a comprehensive data corpus with relevant features, including transcript, genre, title, description, length, and percentage of audio content, which will serve as input for the model development team.



Data Sources:

Data was collected from two primary sources: YouTube channel videos and podcasts, ensuring a diverse set of content types for the blind-friendly video recommendation app.

Pre-processing:

The pre-processing phase involves several crucial steps to prepare the data for model training and testing. The primary tasks include:

Language Constraint:

Data has been restricted to English language videos to maintain a consistent user experience.

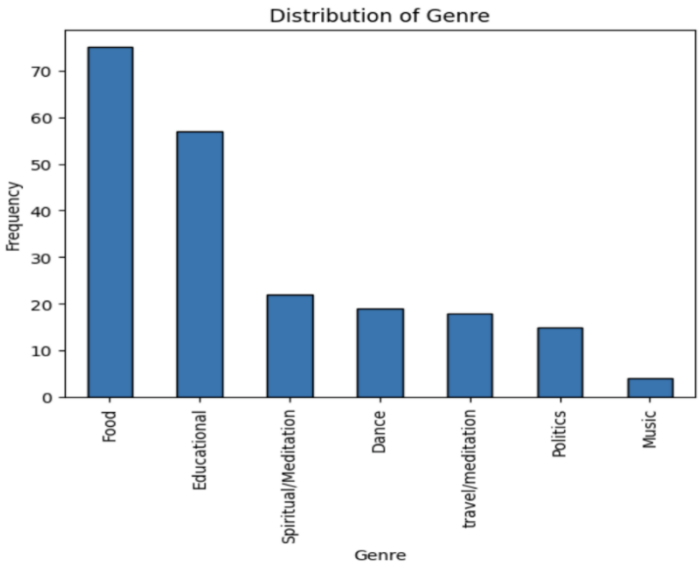
Data Extraction:

Utilizing the YouTube API in Python, we extracted key features such as the transcript of the video, genre, title, description, length of the video, and the percentage of audio content. This process ensures our dataset is enriched with relevant information for the recommendation model.

Features:

We defined specific features crucial for the blind-friendly video recommendation model:

- **Transcript of the Video**: A textual representation of the video's spoken content, facilitating a more in-depth analysis for the recommendation model.
- **Genre of the Video**: Categorizes the video into genres such as politics, education, etc., providing additional context for recommendation.



- **Title:** Provides a concise overview, playing a crucial role in understanding the video's content.
- **Description:** Extracted from the video metadata, offering additional context to enhance understanding.
- **Length of the Video:** The duration influences the user's decision to engage with the content.
- **Percentage of Audio Content:** Quantifies the amount of auditory information present in the video.

Data Corpus:

A structured JSON file contains all extracted features for each video in our dataset. This file serves as the foundation for training and testing the recommendation model.

Labelling:

Videos are categorized as 'positive' or 'negative' based on the extracted features, crucial for supervised learning and making informed recommendations.

Next Steps:

With the data corpus, including the transcript and genre information, our focus shifts to handing over this dataset to the model development team. We anticipate that the inclusion of these will further enhance the model's ability to make accurate and relevant recommendations for visually impaired users.

Summary of "What Makes Videos Accessible to Blind and Visually Impaired People?"

The research paper "What Makes Videos Accessible to Blind and Visually Impaired People?" focuses on the accessibility of online videos for blind and visually impaired (BVI) individuals. The authors conducted formative studies with BVI users, who highlighted the challenges they face in finding accessible videos and identified video accessibility heuristics. The identified heuristics were used to create automated metrics to assess video accessibility. A dataset of accessibility ratings by BVI individuals was collected, and the study found that the automatic video accessibility metrics correlated with the accessibility ratings.
The authors augmented a video search interface with video accessibility metrics and conducted a user study with BVI participants. The results showed that integrating video accessibility metrics into the video search

interface helped BVI users select accessible videos more efficiently, reducing the trial-and-error process. The participants unanimously preferred the augmented interface with accessibility metrics and found it to provide transparency and extra explanations, enabling them to avoid inaccessible videos. The study also demonstrated that the automatic predictions aligned with participants' perceptions of accessibility.

Overall, the research contributes to understanding the accessibility needs of BVI individuals when searching for online videos, identifying video accessibility heuristics, and developing automated metrics to assess video accessibility. The study found a correlation between automatic accessibility metrics and BVI users' perceived video accessibility, providing potential for improving the accessibility of online videos for BVI individuals.

The research paper focuses on the accessibility of online videos for blind and visually impaired (BVI) individuals. The researchers conducted formative studies with BVI YouTube users to understand video accessibility preferences. They identified video accessibility heuristics and implemented automated metrics to assess video accessibility. The study found a correlation between the automatic accessibility metrics and BVI ratings, demonstrating the usefulness of these metrics for BVI individuals when searching for videos.

The paper emphasizes the importance of prioritizing content for description, highlighting the potential to enhance video search by identifying videos with high-quality built-in descriptions. The study also addresses the difference between perceived and true accessibility and outlines the need for further research to understand these disparities.

The researchers selected a sample of videos from the YouTube trending page for their analysis and user study. While the sample size was small, the study obtained expert accessibility ratings from BVI YouTube users and revealed that the sampled videos were more accessible than inaccessible. The researchers acknowledged the need for a larger and more diverse dataset to improve analysis and predictions.

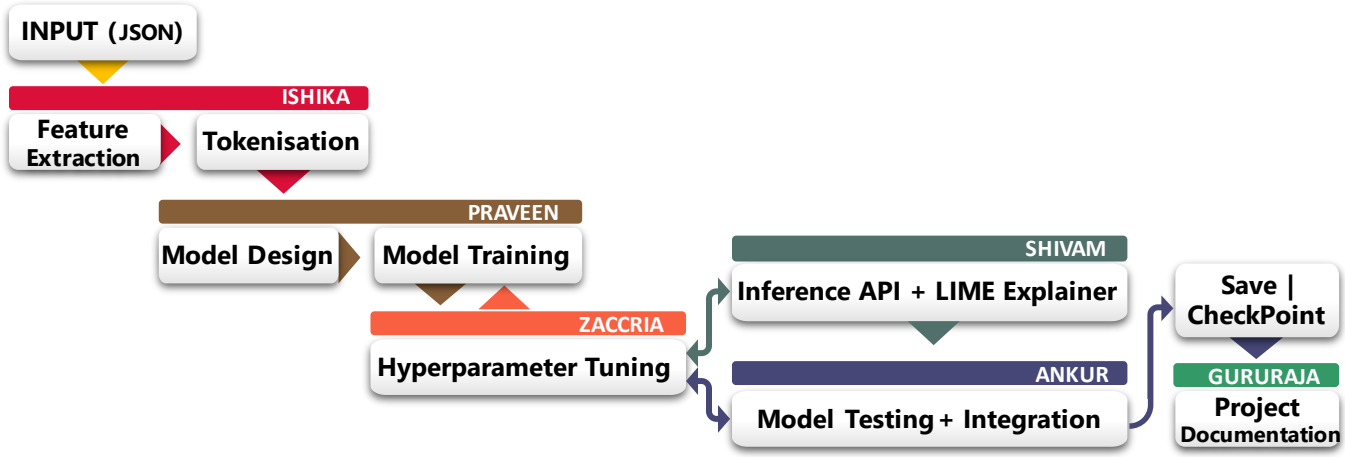
The impact of integrating video accessibility metrics into a video search interface was studied, showing that first-time users experienced efficiency gains and unanimously preferred the new tool. The paper also emphasizes the importance of platform support and scalability, suggesting that video hosting platforms should enable authors to upload audio descriptions and implement the ability for users to filter and browse videos by their accessibility.

In conclusion, the study suggests that surfacing accessible videos on online platforms is a time-consuming burden for BVI users, and the combination of video accessibility heuristics, automated accessibility metrics, and an augmented video interface can systematically and scalably improve the accessibility of video platforms for all users.

Team – 02: Blind-Friendly Video Recommendation App

The objective of this project is to create an application that assists individuals with visual impairments by suggesting videos specifically designed to be accessible for the blind. Group 02 is majorly accountable for tasks such as designing the model, training the dataset, and conducting model testing to the dataset provided by Group 01. We have gathered a comprehensive dataset consisting of various features such as transcripts, genres, titles, descriptions, lengths, and the proportion of audio content. Our initial objective involves preprocessing the collected data through Feature Extraction in required format to the model. Below is a detailed flowchart outlining the different tasks and the assigned individuals responsible for each task.

TASK ALLOCATION:



Libraries utilized in the project

Libraries and Their Purpose:

- **pandas:** Data manipulation.
- **transformers:** DistilBERT model and tokenizer.
- **torch.utils.data:** DataLoader and TensorDataset for efficient training.
- **torch:** General PyTorch functionalities.
- **youtube_transcript_api:** Retrieving transcripts from YouTube videos.
- **re:** Regular expressions for text preprocessing.
- **numpy:** Numerical operations.
- **lime:** Local interpretable model-agnostic explanations for model predictions

PROJECT EXPLANATION

Stage 1: Problem Definition

- **Understanding the Objective:** Develop an app for visually impaired users to navigate and explore YouTube content through text and audio descriptions.
- **Challenges:** Extracting meaningful information from YouTube videos for text and audio summaries.

Stage 2: Data Collection

- **Data Pre-Processing:** Extracted video details, captions, and audio features.

Stage 3: Data Processing and Cleaning

- **Steps:**
    1. Merged video metadata, captions, and audio features.
    2. Removed utilised columns.
    3. Handled missing data and duplicates.

Stage 4: Text and Audio Feature Extraction

- **Text Features:**
    - Used DistilBERT model to extract embeddings from video transcripts.
- **Audio Features:**
    - Extracted audio features using pre-trained models.

Stage 5: Sentiment Analysis

- **Method:** Trained a DistilBERT model for sentiment analysis on video transcripts.
- **Evaluation:** Achieved high accuracy on sentiment classification.

Stage 6: Model Deployment

- **Environment:** Utilized Flask for web app development.
- **Features:** Users can input a YouTube video URL, and the app provides sentiment analysis and audio descriptions.

Stage 7: User Interface Enhancement

- **Additions:** Implemented user-friendly features like audio controls, text summaries, and sentiment indicators.
- **Accessibility:** Ensured compatibility with screen readers.

Stage 8: Model Improvement

- **Fine-tuning:** Extended the model with additional training on custom datasets.
- **Evaluation:** Improved sentiment analysis accuracy.

Stage 9: Code Extensions

- **Libraries Added:**
  - **pandas**: Data manipulation.
  - **transformers**: DistilBERT model and tokenizer.
  - **torch.utils.data**: DataLoader and TensorDataset.
  - **youtube_transcript_api**: YouTube transcript retrieval.
  - **lime**: Local interpretable model-agnostic explanations.
- **Functionalities Added:**
  - Loading a pre-trained model and tokenizer.
  - Extracting video transcript and preprocessing text.
  - Using LIME for explaining model predictions.

Stage 10: LIME Explanation with example

- **Code Sections:**
  - Loading model and tokenizer.
  - Obtaining and preprocessing video transcript.
  - Using LIME for explanation.
- **Explanation Example:**
  - Applied LIME to explain sentiment predictions on the entire transcript.
  - Displayed the explanation using a notebook.

Model Card of Distilbert base (uncased):

This model is a distilled version of the BERT base model. It was introduced in this paper. The code for the distillation process can be found here. This model is uncased: it does not make a difference between english and English.

MODEL DESCRIPTION

DistilBERT is a transformers model, smaller and faster than BERT, which was pretrained on the same corpus in a self-supervised fashion, using the BERT base model as a teacher. This means it was pretrained on the raw texts only, with no humans labelling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts using the BERT base model. More precisely, it was pretrained with three objectives:

Distillation loss: the model was trained to return the same probabilities as the BERT base model.

Masked language modeling (MLM): this is part of the original training loss of the BERT base model. When taking a sentence, the model randomly masks 15% of the words in the input then run the entire masked

sentence through the model and has to predict the masked words. This is different from traditional recurrent neural networks (RNNs) that usually see the words one after the other, or from autoregressive models like GPT which internally mask the future tokens. It allows the model to learn a bidirectional representation of the sentence.

Cosine embedding loss: the model was also trained to generate hidden states as close as possible as the BERT base model.
This way, the model learns the same inner representation of the English language than its teacher model, while being faster for inference or downstream tasks.

Intended uses & limitations

We can use the raw model for either masked language modeling or next sentence prediction, but it's mostly intended to be fine-tuned on a downstream task.
Note that this model is primarily aimed at being fine-tuned on tasks that use the whole sentence (potentially masked) to make decisions, such as sequence classification, token classification or question answering. For tasks such as text generation you should look at model like GPT2.

Limitations and bias

Even if the training data used for this model could be characterized as fairly neutral, this model can have biased predictions. It also inherits some of the bias of its teacher model.
This bias will also affect all fine-tuned versions of this model.

TRAINING DATA

DistilBERT pretrained on the same data as BERT, which is BookCorpus, a dataset consisting of 11,038 unpublished books and English Wikipedia (excluding lists, tables and headers).

Training procedure

PREPROCESSING

The texts are lowercased and tokenized using WordPiece and a vocabulary size of 30,000. The inputs of the model are then of the form:

```
[CLS] Sentence A [SEP] Sentence B [SEP]
```

With probability 0.5, sentence A and sentence B correspond to two consecutive sentences in the original corpus and in the other cases, it's another random sentence in the corpus. Note that what is considered a sentence here is a consecutive span of text usually longer than a single sentence. The only constrain is that the result with the two "sentences" has a combined length of less than 512 tokens.
The details of the masking procedure for each sentence are the following:

- 15% of the tokens are masked. In 80% of the cases, the masked tokens are replaced by `[MASK]`.
- In 10% of the cases, the masked tokens are replaced by a random token (different) from the one they replace. In the 10% remaining cases, the masked tokens are left as is.

PRETRAINING

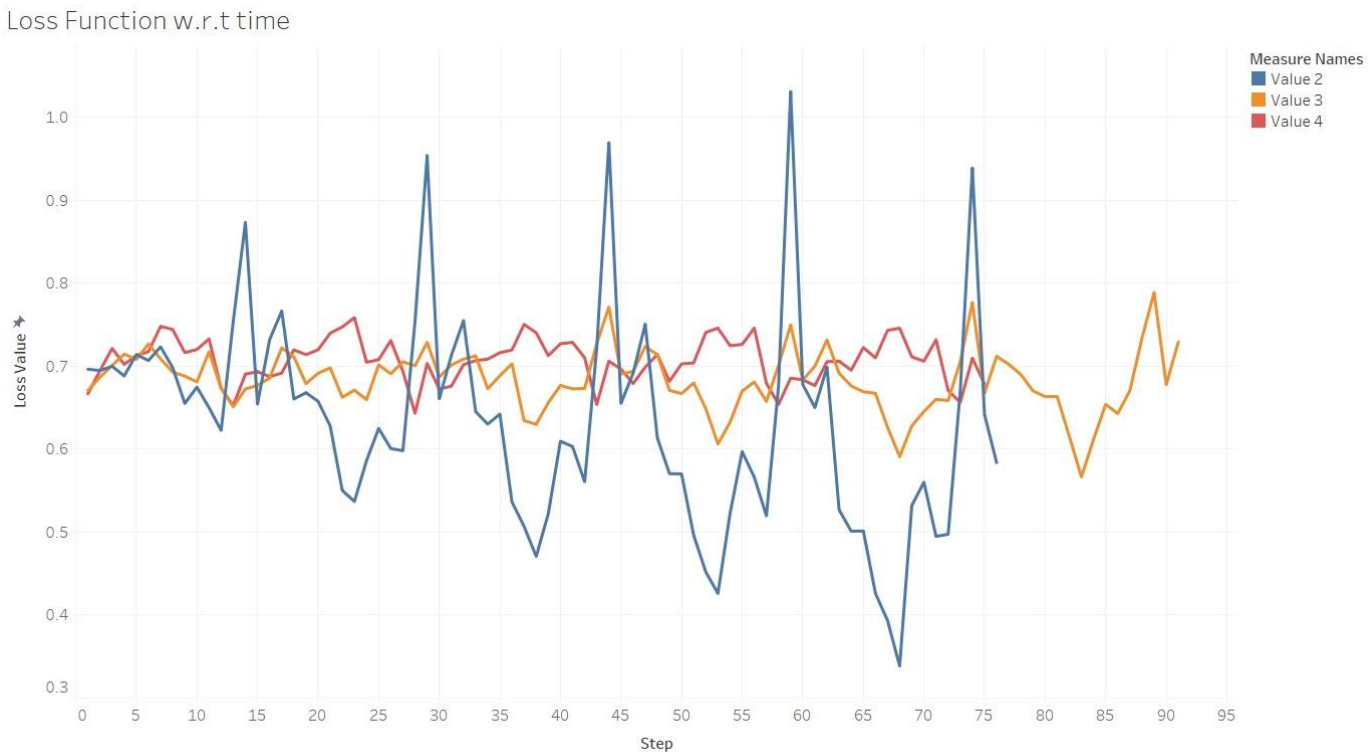
The model was trained on 8 16 GB V100 for 90 hours. See the training code for all hyperparameters details.

Evaluation Trained Model Results

When fine-tuned on downstream tasks, this model achieves the following results:
Glue test results:

Task	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
	82.2	88.5	89.2	91.3	51.3	85.8	87.5	59.9

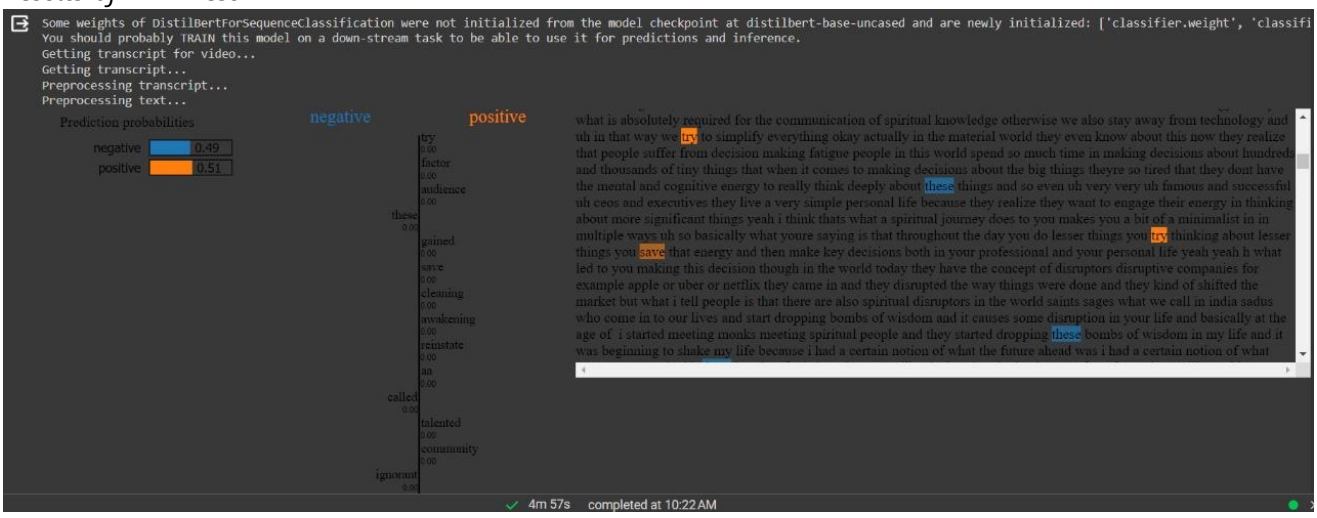
Results of the Model:



The trends of Value 2, Value 3 and Value 4 for Step. Colour shows details about Value 2, Value 3 and Value 4.

Model Name	Learning Rate	Epoch	Batch Size	Learning Rate Scheduler	Data Set	Log file name	Average Loss	Accuracy	Training Time
distilbert_1_model	1.00E-05	5	10	optimizer, step_size=1, gamma=0.9	Old	distilbert_1_model_log		0.666667	45 mins
distilbert_new_2_model	1.00E-05	5	10	optimizer, step_size=1, gamma=0.9	New Trained	distilbert_new_2_model_log	0.633556992	0.6	1 hour 3 mins
distilbert_new_3_model	1.00E-06	5	10	optimizer, step_size=1, gamma=0.9	New Trained	distilbert_new_3_model_log	0.683071379		56 Mins
distilbert_new_4_model	1.00E-07	5	10	optimizer, step_size=1, gamma=0.9	New Trained	distilbert_new_4_model_log	0.706939258	0.666667	1 hour 14 mins

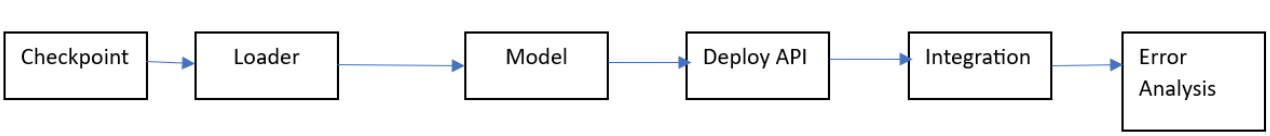
Results of LIME Test:



Team 3 - Model Deployment and API

Executive Summary

- We deployed a machine learning model to a cloud platform and created an API that ranks YouTube videos for accessibility to visually impaired users.
- We collaborated with other teams on data collection, model development and user interface creation.
- We performed error analysis and found patterns in the model's output



1. Checkpoint

- We used transformers & pytorch callback to save the model's state

2. Batch Loader

- We loaded and preprocessed the annotated data using techniques such as tokenization.
- We collaborated with Team 1 (Meera) on data preprocessing.

3. Model

- We collaborated with Team 2 (Ishika) on model architecture and training.

4. API & Deployment

- We deployed the model to a cloud platform Azure using tools such as Docker.
- We developed an API that adheres to industry best practices and tested its robustness and reliability.
- We used various tools and methods for testing, such as Postman.

Cloud Output Link http://pssm318.azurewebsites.net/rerank/?query=nlp&count=2

Testing - docker run -p 8040:5500 testfastapiarticle.azurecr.io/testfastapi2:v2

http://127.0.0.1:8040/rerank/?query=Biryani&count=5

5. Batched Inference

- Batched inference refers to the process of making predictions or inferences on multiple input samples simultaneously, rather than one at a time. In the context of machine learning models, including deep learning models, this means processing a batch of input data together as opposed to individual data points.

6. Integration

- We have received a query from team 4 for scoring. The batch loader returns a list of scores to the same team for their further aggregation. Now we build a docker file which containerizes the files and creates a docker image of the same.
- Then on the Azure server an environment is created and this docker image is uploaded and extracted. This enables any individual or group to access the files on any operationable device and get the desired output.

8.Error Analysis

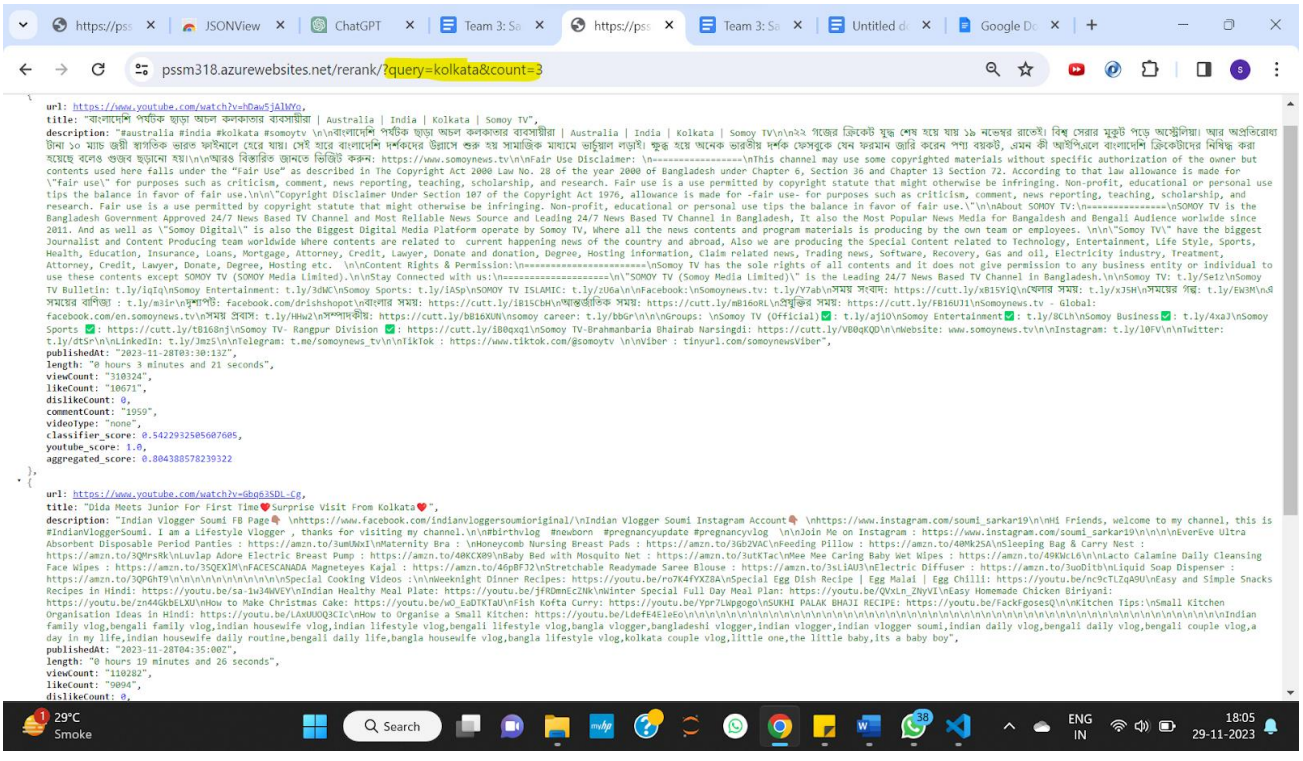
- We performed error analysis and observed that:
    1. We've identified that our model necessitates additional hyperparameter tuning.
    2. The model gave similar scores to songs and videos having the keyword 'blind' in title.
    3. The scores exhibited a limited range between 0.50 and 0.55, indicating the necessity for additional features to enhance our model's training for improved accuracy.

7. Evaluation and Future Steps

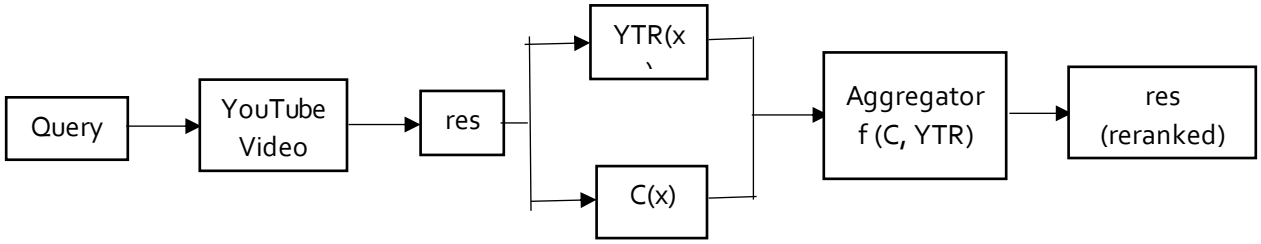
- We evaluated the model and the API using key performance indicators and found the need for further investigation into the model's output consistency.
- We plan to monitor, retrain, and improve the model and the API based on user feedback and evolving needs.

Conclusion

- We successfully deployed a machine learning model for YouTube video ranking and created an API for it.
- We collaborated with other teams and achieved a comprehensive solution.
- We found intriguing findings in error analysis and highlighted the importance of ongoing scrutiny and improvement.



Team 04 : Search and Re-ranking



YouTube API: The code makes use of the YouTube Data API to look for videos matching a search query. It gets key video details like title, description, publish date, view count, etc. An API key is used for authentication. A search YouTube function is defined to run the search. This function fetches a specified number of results and puts relevant video info into a structured format. The example shows a search for NLP tutorial videos with a max of 10 results, displaying the collected video details in easy-to-read JSON. The main capability is extracting and organizing video data, which could enable more analysis or processing. However, it's important to have error handling, check input parameters, and follow API rules for smooth running and compliance.

YouTube Video No.	YouTube Score (a) = $\frac{\text{Total Videos} + 1 - \text{YT Rank}}{\text{Total Videos}}$	Classifier Score (b)	Function Aggregator $\frac{\sqrt{a^2 + b^2}}{\sqrt{2}}$
1	1.0	0.51	0.794
2	0.9	0.44	0.708
3	0.8	0.26	0.595
4	0.7	0.78	0.741
5	0.6	0.11	0.431
6	0.5	0.91	0.734
7	0.4	0.84	0.658
8	0.3	1.00	0.738
9	0.2	0.38	0.304
10	0.1	0.62	0.444

Get Classifier Score: It opens a JSON file and loads its content, extracting the score corresponding to the provided URL. The extracted score is returned as the output.

Get YouTube Score: Get YouTube Score gives scores to videos based on where they are in the YouTube results list. It goes through a list of videos one by one. With each video it checks, it gives a lower score than the video before it. The videos get assigned a score inversely proportional to its position in the list, relative to the total number of videos. The output of this function is a dictionary with keys being the URLs, and the values being the scores.

Get Aggregated Score: Get Aggregated Score function combines both the Classifier and YouTube scores. It squares each of the scores, sums them and then gets the square root of the sum. This follows Pythagoras' theorem to get the length of the vectors described by the two scores if the scores were values on the y-

axis and x-axis. This aggregated score aims to represent a combination of both the classifier and YouTube ranking.

We also considered 2 other methods to combine the two scores to create an aggregated score. What's important to keep in mind is that is that the measure of how well an aggregated score ranks and orders videos is a function of what underlying philosophy we're trying to follow when ranking the videos – do we want to give maximum weight to the accessibility feature and compromise on the relevance of the video (as suggested by YouTube) or vice versa? Or do we want to value them both together the same amount and simply add up the two values to create the aggregated values? As a team we decided that we would like to use an approach that would penalize videos that have a large difference between the two scores – i.e. videos that have an excellent accessibility ranking but very poor relevance ranking, or vice versa. The measure that encompasses this is the L2 norm. We written a code to test this (We've attached the .py file of the code that we wrote to test) We created a list of 10 fictitious videos - (a,b,c,d,e,f,g,h,i,j). We hard coded the relevance score in descending order and accessibility score in ascending order.

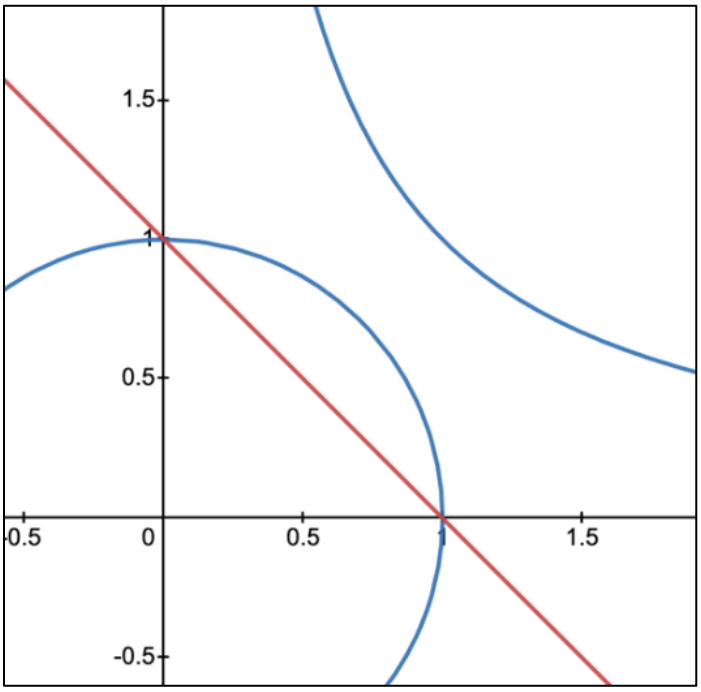
relevance_score = [1, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]
accessibilty_score = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

According to this ranking the middle videos should be preferred since the difference between their scores are low and the extreme videos should be penalized since the have extreme difference in values. We ran the program and saw exactly this. The following are the results.

List of videos = ["a","b","c","d","e","f","g","h","i","j"]

Addition = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
Multiplication= ['a', 'j', 'b', 'i', 'c', 'h', 'd', 'g', 'e', 'f']
L2 Norm = ['e', 'f', 'd', 'g', 'c', 'h', 'b', 'i', 'a', 'j']

Above, we've color coded the videos based on their position in the list of videos and the difference between their scores. The extreme videos have high difference, and the middle ones have low difference. The L2 norm successfully manages to prioritize the green videos. We've also provided a graph below of the 3 algorithms. The circle graph is the L2 norm, the red straight line is simple addition, and the blue descending slope is the simple multiplication. Matching with the results above, the multiplication graph has high Euclidean distance for extreme values, and low distance for relatively short difference in the values. This is why the red comes first in the multiplication list. The reverse is true for the circle and the line is indifferent.



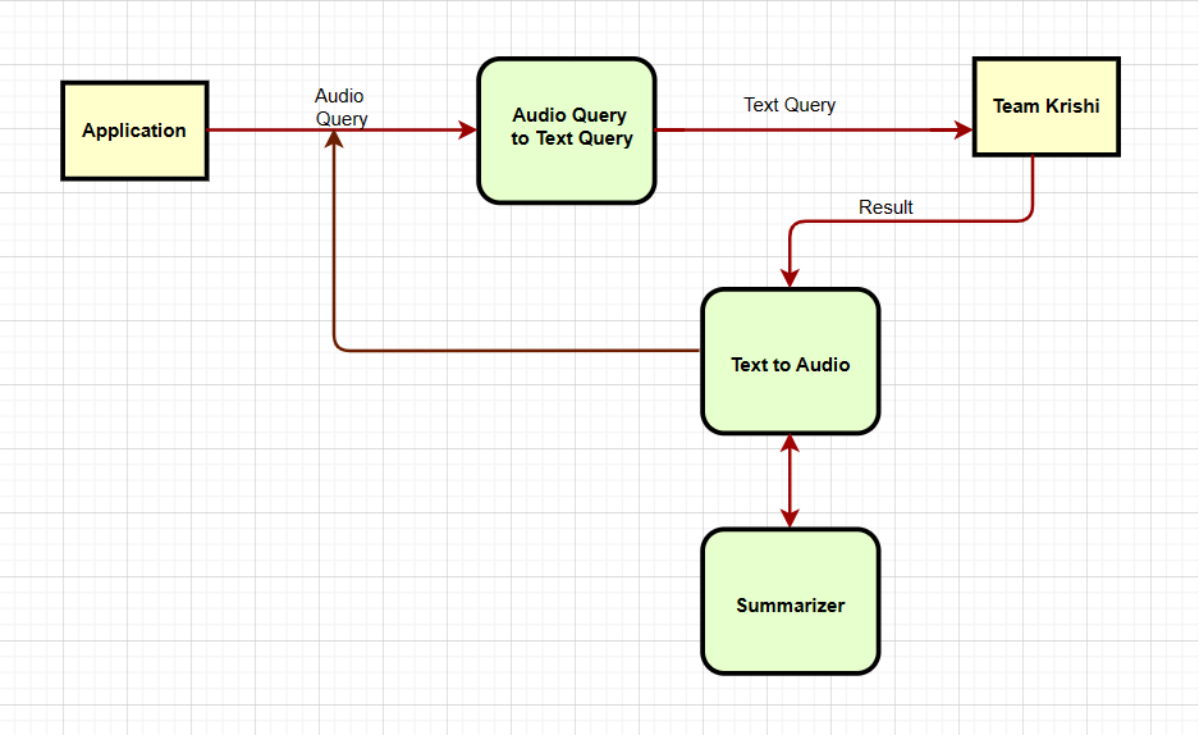
Re-ranking: The function starts a YouTube search, retrieves videos based on the query, processes these videos by calculating aggregated scores, orders the videos by highest to lowest score, and saves the sorted video data to a JSON file called the reranked videos.

Integration, Testing: We receive a JSON file from the deployment and testing team. This file consists of links and their corresponding classifier scores. We use the score and we use the YouTube ranking score to run our aggregator function and we produce a JSON output that consists of a video's metadata, its YouTube ranking score, its classifier score, and its aggregator score. We sent this file to the app team.

Future Scope: In future versions, we could implement a human feedback system in the code that would help enhance search results based on the classifier score.

Team 05: Video Streaming Application for Visually Impaired

Task Delegation:



Major Steps:

1. **Audio to Text:** User will give an audio input to the website which will be then converted to text. When a user gives an audio input, an event will be triggered which will convert this audio to text and this converted text will then be shared with the Krishi's team in the Json format.

```python
import speech_recognition as sr


def audio_to_text()-> str:
    """
    Function to record audio
    """
    r = sr.Recognizer()
    try:
        with sr.Microphone() as source:
            # Reduce the background noise
            r.adjust_for_ambient_noise(source, duration=0.5)

            # listens to users input
            audio = r.listen(source, phrase_time_limit=5)
```

2. **Text to Audio:** The output of Krishi's team will be given in Json file. This Json file will contain the URL, title, and length of the video. This text to audio module will trigger an event which will read the title and duration of the video which are given in the Json file.

```python
#to convert text to audio
import pyttsx3
```

3. **Summarizer:** The text to audio module calls the summarizer passing the video description to it and receives a short ssummary of the video in return. It uses the NLTK module to create a word frequency list.

```python
import bs4 as bs
import urllib.request as url
import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.probability import FreqDist
import heapq
from string import punctuation
```

4. **Web Interface:** The web interface is made on stream lit. It will take the input in the form of audio and will display a list of YouTube videos. This video's title and duration will then be read out to the user and then the user will give a click input to run any video. Space button can be used to start capturing the audio. Number buttons can be used to give the input as to which video can be played.

5. **Integration:** Json output file containing meta data for e.g. title, number of likes, dislikes in which videos are in a ranked manner. This file was integrated with the functions below. The other two modules i.e. text to speech and audio to text were integrated with our front-end website made on steam lit.

6. **Testing:** Testing is done for Audio Query to Text conversion and Text to Audio Conversion. A separate report for the test is attached along with a report showing the output of the various test cases and edge cases.

Libraries used:

1. **streamlit**: Streamlit is an open-source Python library that makes it easy to create and share custom web apps for machine learning and data science.

2. **googleapiclient**: The Google API client library for Python is used to access Google APIs.

3. **speech_recognition**: This is a library for performing speech recognition in Python. It supports multiple speech recognition engines and APIs, including Google Speech Recognition API.

4. **youtube_transcript_api**: This library is used to fetch the transcript of a YouTube video. In our code, it's used in the get_transcript() function to get the transcript of a given YouTube video.

5. **Pytorch**: PyTorch is a machine learning library used for applications such as computer vision and NLP. In our code, it's used to load the model state dictionary, move the inputs and model to the GPU if available, and calculate the softmax probabilities in the get_score() function.

Testing Results of Application Development

Types of Testing:

1. Audio Query to Text Query
2. Text to Audio [Video Title Readout]

Errors Faced:

- Throwing error at the first instance [Unhandled local exception error] - Works after refreshing the page [Not able to find the exact error]- Used try and except in app.py printing "Audio not recognized".
- "No transcript found for en lang" error - Using try and except in utili.py to print "Most likely no transcript was found."
- Summary of YouTube videos failing for vernacular languages

Independent Audio Query to Text Query:

Audio Query to Text Query testing:

Sr. No.	Audio Input	Text Query Output
1	No Audio	Audio Not Recognized
2	Clean Audio-1	Capturing properly and returning list of videos

Sr. No.	Audio Input	Text Query Output
3	Clean Audio-2	Capturing properly and returning list of videos
4	Clean Audio-3	Capturing properly and returning list of videos
5	Clean Audio-4-Language 1-Hindi	Capturing properly and returning list of videos
6	Clean Audio-5-langauge 2- Marathi	Capturing properly and returning list of videos
7	Child Voice-1-Female	Capturing properly and returning list of videos
8	Child Voice-2-male	Capturing properly and returning list of videos
9	Feeble Audio	Capturing properly but not returning the correct list of videos
10	Very Loud Audio	Capturing properly and returning list of videos
11	Audio with traffic sound	Capturing properly and returning list of videos
12	Random Voice	Audio not Recognized
13	Random Non-Stop Conversations	Audio not Recognized
14	Audio with background voice	Capturing properly and returning list of videos

Total number of trials: **42.**
Number of successful retrieval and display of list: **35.**
Number of times, error was displayed: **7.**
Percentage of successful trials: 35/42*100 = **83.33%**

Independent Text to Audio [Video Title Readout]

Version 1:

Test Scenario 1: Passed

Enter 0 to hear the next video details. Enter any other key to check this video.
The speech is properly reading the title as well as the length.
Input: 0
Output: The function is reading the next YouTube video and title.
Expected Output: Next YouTube video and title to be read.

Test Scenario 2: Passed

Enter 0 to hear the next video details. Enter any other key check this video.
The speech is properly reading the title as well as the length.
Input: 1
Output: The function is opening the URL as specified.
Expected Output: URL of current video to open.

Test Scenario 3: Passed

When no details are passed in the json file
Output: Sorry! No result found.
Expected Output: To convey no output found.

Test Scenario 4: Passed

When there is no next video present.

Output: "Enter o to hear the next video details. Enter any other key to check this video. But there is no video after this one."
Expected Output: The user should be informed that there is no next video.
Total number of trials: **12**
Total number of correct outcomes: **12**
Total number of incorrect outcomes: **0**
Percentage of successful trials 12/12 * 100= **100%**

Version 2:

Test Scenario 1: Passed
Scenario: Enter A to hear the previous video details.
Input: A
Output: The function is reading the previous YouTube video and title.
Expected Output: Previous YouTube video and title to be read.

Test Scenario 2: Passed
Scenario: Enter D to hear the next video details.
Input: D
Output: The function is reading the next YouTube video and title.
Expected Output: Previous YouTube video and title to be read.

Test Scenario 3: Passed
Scenario: Enter S to select the current video.
Input: S
Output: Current Video is playing on YouTube.
Expected Output: Current video to play on YouTube.

Test Scenario 4: Passed
Scenario: Enter W to exit.
Input: W
Output: The user is informed about exiting.
Expected Output: The user should be informed that the user has decided to exit the application.

Test Scenario 5: Passed
Scenario: Any other key is pressed instead of A, W, S, D.
Input: M
Output: The user is informed that he has selected wrong input and kindly select any one of A, W, S, D.
Expected Output: The user should be informed that the user has decided to exit the application.

Test Scenario 6: Passed
Scenario: Previous Video option (A) is selected at the first option.
Input: A
Output: The user is informed that it is an invalid video. Kindly select any one of A, W, S or D.

Expected Output: The user is informed that he has selected wrong input and kindly select any one of A, W, S or D.

Test Scenario 7: Failed

Scenario: Next Video option (D) is selected at the last option.
Input: D
Output: Previous video details are read.
Expected Output: The user should be informed that this is the last video.

Total number of trials: **21**
Total number of correct outcomes: **18**
Total number of incorrect outcomes: **3**
Percentage of successful trials 18/21 * 100= **85.71%**

--o--