Presentation notes

Problems and opportunities

- search for which prob we can solve... then my teammate was in dnfc and he expressed the need the club has.
- we interacted and realised the potential of using technology for content generation and content analysis

Objectives

1. *read out the obj 1 :* building a whole website was a big and time consuming task. As it would require us to devlop more exterise and explore newer technologies. and given our time and skill contraint we decided to keep the goal realistic
2. we realised that the recent developments in genAI and LLM are quite useful in domain where natural language related task are carried. and since a lot of the work of DNFC revolved around text like writing scripts, creating dialogues, analyzing emotions and sentiments from paragraph, etc we decide to find innovative solution in this domain

Overview

- just a quick overview of what all we have created. we'll dive in deeper one by one

Flask app

1. When the user first hit out website he'll come on the landing page made by using.....

2. he can navigate through various pages of our websites like contact us, view the short film, about us....
3. if he wants to access features to be used only after login thus for user management and authentication...

OOP

1. we hadin over...
2. there wasnt much possible to show OOP being being explicitly applied to frontend part but i can explan about the backend wherei we used python.
3. One of the many principles in use was ..... encapsuated the data and the related functions into separate file according to their types like a module for views, authentication, flask app initialization, etc
4. another important concept of abstraction can be understood via the python packages. by importing them we abstracted away the complexity involed in creating a particular tasks like defining routes, handling req, interaction with database. thus by simply having the knowledge of classes and object available in that library we can effectively use them to solve our problems.
5. inheritance can be obsered via the library Flask we imported and there is derived class named create_app which inherits all the attributes and method and also has its own to create an instance of flask application, which can be used to interact with its properties and method
6. also the flask_alche.. gives the power to manage the database in OOP way by creating classes representing models for our database.

Comment Analysis

1. if a user want to analyze comments in a much more detailed way he can use our feature
2. on the comments we provide sentiment analysis, sarcasm detection and constructive feedback summary
3. Under the hood..
    a. THe user puts the link
    b. we fetch 50 comment using api

c. preprocess the comment to separate into english and marthi texts using re.

d. pass the english comments to LLM for sentimenet analysis

e. similarly for marathi language specific LLM for marathi comments

f. visualize the sentiment analysi via pie chart

g. aslo provide a word cloud for showing the most frequently used words

    i. wordcloud library

    ii. and plotted the image using matplotlib

h. Sarcasm detection i make using langchain for creating prompt templates along with google gemini LLM

    a. prompt template are used to craft and send specific prompts to the llm so as to direct its response in dersired way.

    b. thus i sent the list of comment and instructed what to look for in sarcastic comments eg out of context comments, exggarated comments.

    c. thus accordingly i find the sarcastic comments

i. in sarcastic comments i try to pick out comment in which genuinely provide feedback

    a. most of the comment are very like great! , nice video!, mast, etc

    b. but in those itself someone had taken effort to provide a genuine feedback

    c. thus given a list of comment and context i again use prompt template to instruct the llm to pick comments based on attribtues like longer than usual comments, in context, etc.

Monologue Generation

1. The club has a collection of monologue (why is monologue) which can be made available to the user who needs.

2. we couldve simply made avalible all the monologue scripts as it is but it would be quite tedious task for a user to find the right monologue as sometimes the right filter for searching available
3. the user can express what he want in the normal way as if he's talking to his friend about what he wants and out feature understand and provide the most relevant monolgue
4. thus we tackled this issue using embedding and llms
    a. embedding capture the meaning of a given text. we can make sense of a simple sentence as it is but computers can only understand numbers thus the meaning of the sentence in numerical format is not but embedding.
    b. thus i convert all the monologue into embedding and store into vectorstore, its like a database to store numbers.
    c. thus the query of the user is also converted to embeddings
    d. now we compare the embedding created by the user query with the embedding in vectoretore using a technique called cosine similarity which help find the most matching set of embedding.
        i. thus i pass all these embedding and craft response in required format using prompt template.

Script Analysis

1. Drama and film club is all about scripts sometimes of film, sometimes of drama. thus to enable quicker analysis of the scripts i built this feature.
2. one need to upload the script in pdf format and the program will read the text and convert to embedding.
3. here the process differs as to how we are creating the embedding. there embedding were created wrt to each monologue here we chunk together texts in size of around 800 chars and then create its embeddings.
4. then according to the query the most relevant answer is found and rendered via LLM

Potential challenges with LLM applications

1. LLM sometimes produce unreliable results: its an inherit fault that all LLM have.
2. LLM models are huge and require a lot of resources to run: running on limited resources makes them inefficient. they need a lot of ram and CPU-GPU power to perform optimally.
3. high learning curve: need to learn a lot before implemeting
4. **Constantly changing domain, thus frequent changes in support libraries**: new tech thus libraries are being modified quite frequently then always on your toes to learn and change.
5. **Due to lack of available resources , debugging issues and finding help change be time consuming**: thus debugging issues might take time.
6. huge

Future plans.

1. improvise the web app prototype and make it deployment ready.
2. Find alternate way to make deploying LLM application more feasible and efficient.
3. Develop more and newer problem by advancing our knowledge in tis domain

Libraries used

1. **Flask (`from flask import Flask`):**
    - Flask is a micro web framework for Python. It allows you to build web applications easily and quickly. In this code, it is used to create the Flask application instance `app`.
2. **Flask-SQLAlchemy (`from flask_sqlalchemy import SQLAlchemy`):**
    - Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy, a powerful relational database toolkit. It simplifies database operations within Flask applications. In this code, it is used to interact with the database.
3. **SQLAlchemy (`from flask_sqlalchemy import SQLAlchemy`):**
    - SQLAlchemy is an Object-Relational Mapping (ORM) library for Python. It provides a high-level interface for interacting with relational databases. In this code, it is used via Flask-SQLAlchemy to define and manipulate database models.
4. **os (`from os import path`):**
    - The `os` module provides a portable way of using operating system-dependent functionality. In this code, it's used to check for the existence of a file path.
5. **Flask-Login (`from flask_login import LoginManager`):**
    - Flask-Login is an extension for Flask that provides user session management, handling user authentication, and more. It simplifies the process of managing user sessions and authentication in Flask applications.
6. **db (`db = SQLAlchemy()`) and `DB_NAME`:**
    - `db` is an instance of the SQLAlchemy class, initialized without a Flask application context. It represents the database connection. `DB_NAME` is a string constant representing the name of the database file
7. **requests (`import requests`):**

- The `requests` library is used for making HTTP requests in Python. In this code, it is utilized to fetch data from the YouTube Data API.

8. **json (`import json`)**:
   - The `json` module provides functions for encoding and decoding JSON data. It is used to work with JSON data in this code.
9. **streamlit (`import streamlit as st`)**:
   - Streamlit is a Python library for creating interactive web applications. It simplifies the process of building data-driven web apps. In this code, it is used for creating the user interface.
10. **os (`import os`)**:
    - The `os` module provides functions for interacting with the operating system. In this code, it is used for accessing environment variables and working with file paths.
11. **langchain_google_genai (`from langchain_google_genai import ChatGoogleGenerativeAI`)**:
    - This appears to be a custom library or package named `langchain_google_genai`, which is used for language processing tasks. Specifically, it imports a class `ChatGoogleGenerativeAI`.
12. **langchain (`from langchain.output_parsers import ResponseSchema`, `from langchain.output_parsers import StructuredOutputParser`, `from langchain.prompts import ChatPromptTemplate`, `from langchain.chains import SequentialChain, LLMChain`)**:
    - This seems to be another custom library or package named `langchain`, used for language processing and text generation tasks. It imports various classes and modules related to output parsing, prompts, and chain models.
13. **transformers (`from transformers import pipeline`)**:
    - The `transformers` library provides state-of-the-art natural language processing (NLP) models. In this code, it is used for loading and using pre-trained NLP models for sentiment analysis.
14. **dotenv (`from dotenv import load_dotenv`)**:
    - The `dotenv` module allows loading environment variables from a `.env` file into the environment. It is used to load sensitive information like API keys from a `.env` file.

15. **re (`import re`)**:
    - The `re` module provides support for regular expressions in Python. It is used in this code for pattern matching to detect Marathi characters in comments.
16. **plotly (`import plotly.express as px`)**:
    - Plotly is a graphing library for creating interactive visualizations. In this code, it is used to generate visualizations such as pie charts.
17. **wordcloud (`from wordcloud import WordCloud`)**:
    - WordCloud is a library for generating word clouds from text data. It is used in this code to create word clouds from comments.
18. **matplotlib (`import matplotlib.pyplot as plt`)**:
    - Matplotlib is a plotting library for Python. In this code, it is used for creating plots and visualizations.

19. **langchain_google_genai (`from langchain_google_genai import ChatGoogleGenerativeAI`)**:
    - This imports the `ChatGoogleGenerativeAI` class from the `langchain_google_genai` module. It seems to be a part of a custom library or package related to language processing and generation tasks.
20. **langchain_openai (`from langchain_openai import OpenAIEmbeddings`)**:
    - This imports the `OpenAIEmbeddings` class from the `langchain_openai` module. It appears to be another part of the custom library or package for handling OpenAI embeddings, which are representations of text data.
21. **langchain_community.vectorstores (`from langchain_community.vectorstores import FAISS`)**:
    - This imports the `FAISS` class from the `langchain_community.vectorstores` module. FAISS (Facebook AI Similarity Search) is a library for efficient similarity search and clustering of dense vectors.
22. **langchain.document_loaders.csv_loader (`from langchain.document_loaders.csv_loader import CSVLoader`)**:

- This imports the `CSVLoader` class from the `langchain.document_loaders.csv_loader` module. It seems to be a part of the custom library for loading data from CSV files.

23. **langchain.prompts (`from langchain.prompts import PromptTemplate`):**
    - This imports the `PromptTemplate` class from the `langchain.prompts` module. It is likely used for defining templates for prompts in language generation tasks.
24. **langchain.chains (`from langchain.chains import RetrievalQA`):**
    - This imports the `RetrievalQA` class from the `langchain.chains` module. It seems to be a part of the custom library for building question-answering systems based on retrieval mechanisms.

25. **PyPDF2 (`from PyPDF2 import PdfReader`):**
    - PyPDF2 is a library for reading and manipulating PDF files in Python. It allows you to extract text, merge, and split PDF documents. In this code, it's used to read the content of uploaded PDF files.
26. **langchain.embeddings.openai (`from langchain.embeddings.openai import OpenAIEmbeddings`):**
    - This imports the `OpenAIEmbeddings` class from the `langchain.embeddings.openai` module. It's part of a custom library for handling OpenAI embeddings, which are representations of text data.
27. **langchain.text_splitter (`from langchain.text_splitter import CharacterTextSplitter`):**
    - This imports the `CharacterTextSplitter` class from the `langchain.text_splitter` module. It's used for splitting text into smaller chunks based on characters, with specified parameters such as chunk size and overlap.
28. **langchain.vectorstores (`from langchain.vectorstores import FAISS`):**
    - This imports the `FAISS` class from the `langchain.vectorstores` module. FAISS is a library for efficient similarity search and clustering of dense vectors. It's used here to create a vector store from text data.

29. **typing_extensions (`from typing_extensions import Concatenate`)**:
    - The `typing_extensions` module provides experimental type hints that are not yet available in the Python standard library. In this code, it seems to import `Concatenate`, but it's not used.
30. **langchain.chains.question_answering (`from langchain.chains.question_answering import load_qa_chain`)**:
    - This imports the `load_qa_chain` function from the `langchain.chains.question_answering` module. It's likely used to load a question-answering chain model.
31. **langchain_openai (`from langchain_openai import OpenAI`)**:
    - This imports the `OpenAI` class from the `langchain_openai` module. It's used as part of the question-answering process, likely for interacting with OpenAI's API.
32. **langchain.callbacks (`from langchain.callbacks import get_openai_callback`)**:
    - This imports the `get_openai_callback` function from the `langchain.callbacks` module. It's used for obtaining a callback function, possibly for monitoring or logging purpose