CSE 2004:

DATABASE MANAGEMENT SYSTEMS PROJECT

REPORT: - VOICE ANALYSED MOOD BASED MUSIC PLAYLIST

MADE BY:

GROUP 12,

RAGHAVENDRA TIWARY, 19BCE0425

SUPRIT DARSAN SHRESTHA,19BCE2584

DEVIREDDY SAI LOHITH REDDY, 19BCE0450

FACULTY:

PRADEEP KUMAR ROY

INDEX:

ACKNOWLEDGEMENT:

We would like to present our heartfelt gratitude to our faculty, Dr. Pradeep Kumar Roy, for his valuable guidance and inputs towards the completion of the project and solving all the doubts which arose during the project. We will forever be grateful to him for his help.

Thank You.

Raghavendra Tiwary

Suprit Darsan Shreshta

Devireddy Sai Lohith Reddy

INTRODUCTION:

Voice Based Mood Analyzed System is named as such as it is a service which takes in user voice tone and analyses this for the mood of the user. The user says something along the lines of “Play a song” and just this line is used to detect the mood of user. This mood is then used to find the tracks suitable for the mood of the listener. This service employs the application of Natural Language Processing, a domain of Artificial Intelligence and Machine Learning, and data signal processing.

Current systems like Google voice assistant and Alexa also employ natural language processing. But their services are limited to just converting the speech to text. The mood detection from speech is not there in these services.

This project is based on implementing a database of music playlist which would be created from a database of songs by using Spotify API and analyzing the lyrics and music of the song, tags provided by the users and the artist. It will use Watson Tone Analyzer AI and NLP for processing the voice and lyrics of the songs. Based on the voice and the tone of the voice, it will automatically curate a playlist of songs for that mood or for the keyword (like pop, rock music, sad songs, EDM, 90s, Party songs, artist) provided by the user. To make the database of playlists, we will be using NLP for analyzing the song lyrics and the music of the songs. Based on the words used in the lyrics, music and the tags provided by the artist uploading the songs, each song/music score will be tagged with some keywords consisting of genre and mood of the song. Whenever a keyword is provided or mood analyzed by tone analyzer is provided, the database will be searched and some songs matching with the tag will be added to a new and empty playlist. If the user reaches the second last song of the playlist, again the database consisting of songs will be searched and more songs matching the tags will be added to the playlist. Also, users can give multiple keywords for which, a playlist will be created in the real time with songs having all the tags mentioned.

Songs and playlists will also be created on the mood basis. Finally, after each new playlist is created on the demands of the user, it will be saved in the database. Moving on from the database and processing aspect of the project, the databases will be accessed via a webpage which will be linked with Watson Tone Analyzer AI and NLP module. Users will be able to securely login to the webpage and ask it to play songs. Artists will also be able to log in to the system and upload their newly created music. They will also be asked to provide tags to the music. It will also show the lyrics of the songs if the user says “Lyrics” keyword by using NLP.

Limitations: This model is currently limited in accurately analyzing the tone of user due to the presence of noise. If there is a noise present in the background, then it falters in accuracy as the waveform which is used for analysis provides with inefficient data for analysis.

Future Work: These limitations can be overcome by further work on data signal processing and working on the aspect of alienating the background noise for the analysis. This can be done by detecting the erroneous and inconsistent highs and lows in the data upon analysis.

Process Flowchart:

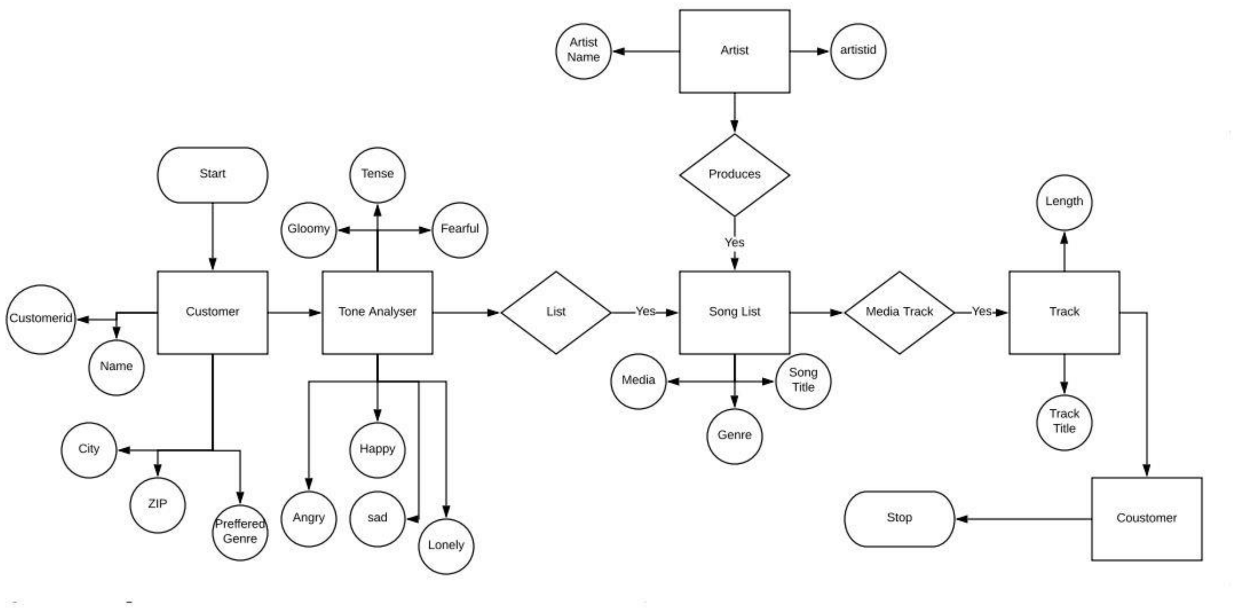


Figure 1:

This image depicts how a customer, an artist and the song list and tracks are interacting with each other. From this image, we can see that Costumer's tone is passed to tone analyser and it passes it to database to create a song list from which songs are played.

ER Diagram:

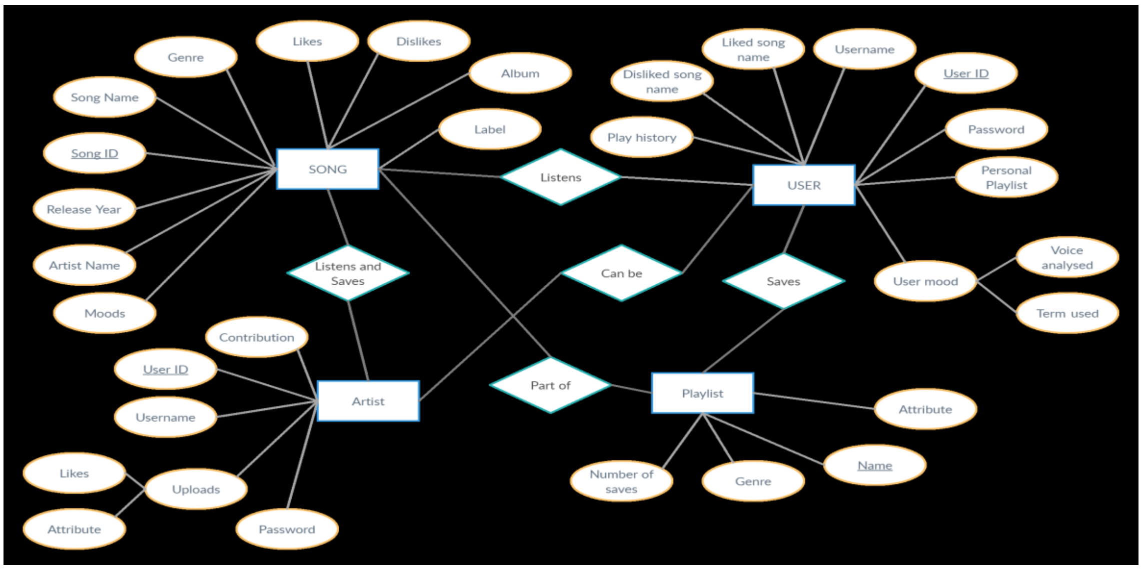


Figure 2: This image shows different entities, namely, song, artist, user, playlist, and their attributes. This also shows the different interactions each entity is having with another and how entities are related.

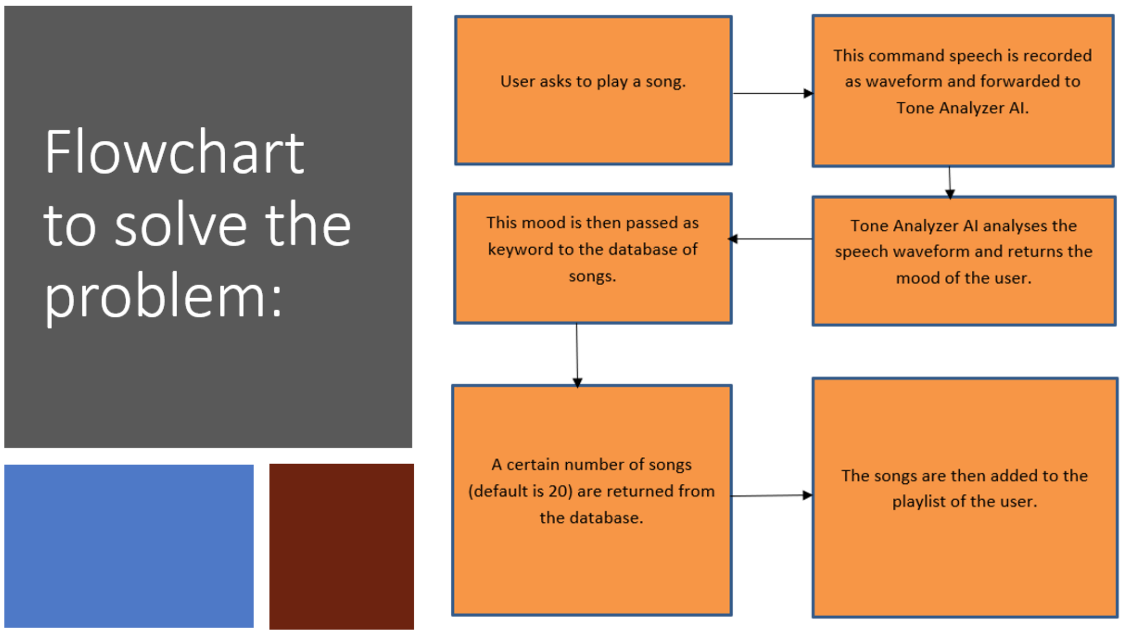

Flowchart to solve the problem:
User asks to play a song.
This command speech is recorded as waveform and forwarded to Tone Analyzer AI.
Tone Analyzer AI analyses the speech waveform and returns the mood of the user.
This mood is then passed as keyword to the database of songs.
A certain number of songs (default is 20) are returned from the database.
The songs are then added to the playlist of the user.

Figure 3: This image shows step by step how the processing is done on the backend of the project. First, the audio of request speech is recorded and saved as waveform. This waveform is passed to tone analyser where it is analysed and a tone is provided. This tone is passed as search parameter to database, and a playlist is created based on the songs identified for that attribute.

Use of Machine Learning:

For our project, we have used the concept of natural language processing. The ML model powering our project is SVM. When we run the project, an audio is recorded which is stored in form of waveforms. This waveform is then passed to the IBM cloud-based Tone analyser, which analyses and provides us with mood of the user.

The service uses linguistic analysis and the correlation between the linguistic features of written text and emotional and language tones to develop scores for each of these tone dimensions

SVM:

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. The data points are generated from the waveform provided.

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes.

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

For the machine-learning model, This model leveraged several categories of features:

- N-gram features
- Lexical features from various dictionaries
- The existence of second-person references in the conversation
- Some dialogue-specific features such as saying thank you or apologizing

The reason for using SVM as compared to other algorithms like Naïve Bayes Algorithm and kNN Algorithm stems from the fact that SVM is used to analyse datasets in hyperplane. As our waveform is plotted in hyperplane, this increases the accuracy to determine the different highs and lows of the waveform. The data points collected from this hyperplane are used to determine the mood of the user. This model provides best accuracy at present as compared to other models, with accuracy over 80 percent as compared to other state of the art models’ 60 percent.

CODE:

Record.py:

```
import sounddevice as sd
import OpenSSL.tsafe
from scipy.io.wavfile import write

def record():
    fs = 44100  # Sample rate
    seconds = 5  # Duration of recording

    myrecording = sd.rec(int(seconds * fs), samplerate=fs, channels=
2)
    sd.wait()  # Wait until recording is finished
    write('myfile.wav', fs, myrecording)  # Save as WAV file
```

Explanation: This code is used to record the command user gives to play the songs. The recorded audio is saved as waveform by using this code.

Speechanalysis.py:

```
import json
import Record
import os
from ibm_watson import ToneAnalyzerV3
from os.path import join, dirname
from ibm_watson import SpeechToTextV1
from ibm_watson.websocket import RecognizeCallback, AudioSource
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator



def speechtext():
    print('Recording Started')
    Record.record()
    if(os.path.exists('myfile.wav')==True):
        print('recording complete')
    authenticator = IAMAuthenticator('PDkEi3h3AkbNqldQGxu8qJVm7JojaJ
2X--xdX1k8b9aj')# Paste the Authorisation token
    service = SpeechToTextV1(authenticator = authenticator)

    #Paste the authorisation url
    service.set_service_url('https://api.eu-gb.speech-to-
text.watson.cloud.ibm.com/instances/dfef8b97-384a-48ed-a446-
969a821a2372')
```

```
    with open(join(dirname('__file__'), r'myfile.wav'), 'rb') as aud
io_file:
        dic = json.loads(
            json.dumps(
                    service.recognize(
                        audio=audio_file,
                        content_type='audio/wav',
                        model='en-US_NarrowbandModel',
                    continuous=True).get_result(), indent=2))

    str = ""

    while bool(dic.get('results')):
        str = dic.get('results').pop().get('alternatives').pop().get
('transcript')+str[:]


    return str




def analysis():
    authenticator = IAMAuthenticator('Bp6npJ89891S23rprtJHyg_Pg_3MMW
YzayU1fLM0SS_j') # Paste the authorisation token
    tone_analyzer = ToneAnalyzerV3(
        version='2017-09-21',
        authenticator=authenticator
    )

    # Paste the authorisation url
    tone_analyzer.set_service_url('https://api.eu-gb.tone-
analyzer.watson.cloud.ibm.com/instances/1077e8e7-9ad4-4449-858d-
b34f77fb1c40')

    text=speechtext()

    tone_analysis = tone_analyzer.tone(
        {'text': text},
        content_type='application/json'
    ).get_result()

    os.remove('myfile.wav')

    try:
        tone=tone_analysis["document_tone"]["tones"][0]["tone_id"]
        #print(tone)
        return tone+' '+text
    except:
        pass
    return text
```

Explanation: The above code is used for the analysis aspect of the project. This code takes the waveform as input and does two things: 1) passes the waveform for analysis. 2) Converts audio to text for detecting keywords like ‘happy’, ‘sad’, etc. It returns the keyword and mood as a result.

spotify_client.py:

```
import random
import requests
import string
import urllib
import speechtotext



class SpotifyClient(object):
    def __init__(self, api_token):
        self.api_token = api_token

    def get_random_tracks(self):
        word1=speechtotext.analysis()
        #print(word1)
        #wildcard = f'%{random.choice(string.ascii_lowercase)}%'
        wildcard = f'%{word1[random.randint(0,len(word1)-1)]}%'
        query = urllib.parse.quote(wildcard)
        offset = random.randint(0, 2000)
        url = f"https://api.spotify.com/v1/search?q={query}&offset={
offset}&type=track"
        response = requests.get(
            url,
            headers={
                "Content-Type": "application/json",
                "Authorization": f"Bearer {self.api_token}"
            }
        )
        response_json = response.json()

        tracks = [
            track for track in response_json['tracks']['items']
        ]

        print(f'Found {len(tracks)} tracks to add to your library')

        return tracks
    def add_tracks_to_library(self, track_ids):
        url = "https://api.spotify.com/v1/me/tracks"
        response = requests.put(
            url,
            json={
                "ids": track_ids
            },
            headers={
                "Content-Type": "application/json",
                "Authorization": f"Bearer {self.api_token}"
            }
        )

        return response.ok
```

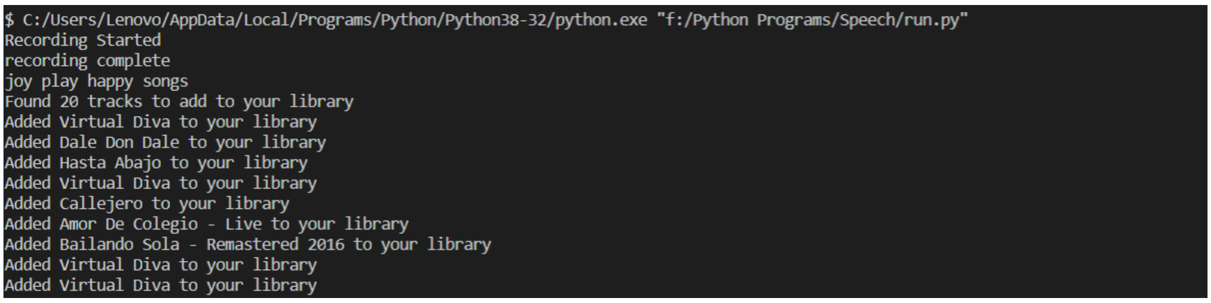
Explanation: This code is utilised for getting the songs from the database by using Spotify API and providing mood and keyword as endpoint parameter. The code returns n (default is 20) number of songs as output.

run.py:

```
import os

from spotify_client import SpotifyClient


def run():
    spotify_client = SpotifyClient(os.getenv('SPOTIFY_AUTH_TOKEN'))
    random_tracks = spotify_client.get_random_tracks()
    track_ids = [track['id'] for track in random_tracks]
    #track_names=[track['name'] for track in random_tracks]


    was_added_to_library = spotify_client.add_tracks_to_library(trac
k_ids)
    if was_added_to_library:
        for track in random_tracks:
            print(f"Added {track['name']} to your library")

    #print(track_names)


if __name__ == '__main__':
    run()
```

Explanation. This code imports the results of all of the codes above and runs them. This the code which adds the returned list of songs to the playlist.

OUTPUT:



```
$ C:/Users/Lenovo/AppData/Local/Programs/Python/Python38-32/python.exe "f:/Python Programs/Speech/run.py"
Recording Started
recording complete
joy play happy songs
Found 20 tracks to add to your library
Added Virtual Diva to your library
Added Dale Don Dale to your library
Added Hasta Abajo to your library
Added Virtual Diva to your library
Added Callejero to your library
Added Amor De Colegio - Live to your library
Added Bailando Sola - Remastered 2016 to your library
Added Virtual Diva to your library
Added Virtual Diva to your library
```

References:

1. A Semantic Approach to Emotion Recognition Using IBM Watson Bluemix tone Analyzer and translator Language https://ieeexplore.ieee.org/abstract/document/9012970
2. Spotify API token: https://developer.spotify.com/console/put-current-user-saved-tracks/?ids=