# MACHINE LEARNING FOR COMMUNICATION SYSTEMS LAB REPORT
# MEIC501P

## TASK 1: PERFORMANCE ANALYSIS OF SUPERVISED AND UNSUPERVISED LEARNING


**NAME:** DURGA SITALAKSHMI S

**REGISTRATION NO:** 24MEC0024

**PHONE NUMBER:** +91 9731850288

**MAIL ID:** durgasitalakshmi.s2024@vitstudent.ac.in

**TASK 1a:** Performance Analysis of Supervised learning

```
#TASK 1(a) : PERFORMANCE ANALYSIS OF SUPERVISED LEARNING
# 24MEC0024
#PDURGA SITALAKSHMI S
#PHONE NUMBER : 9731850288
#MEIC501P

import pandas as pd
# pandas library is imported to do data manipulation and analysis. Making dataframes and changing it as per we want

import numpy as np
# to perform mathematical operations

import librosa
# python library for audio and music analysis. It extracts features from the audio files and process for further use in ML .

from glob import glob
# finds all the pathnames matching a specified pattern

audio_files =glob('Downloads\Actor_01/*.wav')
# collects all .wav extension file from Actor_01 folder from Downloads folder


from matplotlib import pyplot as plt
# Importing pyplot module from matplotlib library for plotting
```

```
lst = []
# creates an empty array to append the audio files

length=len(audio_files)
# finds the length of the audio_files

for i in range (length):
    # the commands inside the for loop is to append the audio files to lst array
    y,sr = librosa.load(audio_files[i])
    # The librosa.load() function is used to load an audio file.
    #It returns two values:
    # 1.y, which is a NumPy array containing the audio time series
    # 2.sr, which is the sampling rate of the audio.
    lst.append(y.max()*1000)
    #y.max() computes the highest amplitude in the audio file.
lst.sort()
# sorts the lst in asencding order default. For reverse/descending order , use lst.sort(reverse=True)

df2=pd.DataFrame(lst)
# here the pandas library is called and used to make a dataframe of lst

df2.columns=["freq"]
# naming the column as freq

df2.to_csv("rec.csv")
# converting the dataframe to csv file and saving it as rec.csv file
```

```
print(df2)
# printing the dataframe
```

| | freq |
|---|---|
| 0 | 22.052493 |
| 1 | 22.440949 |
| 2 | 26.036292 |
| 3 | 26.980430 |
| 4 | 28.652139 |
| 5 | 29.932763 |
| 6 | 31.419680 |
| 7 | 33.317581 |
| 8 | 35.650913 |
| 9 | 35.884488 |
| 10 | 36.971465 |
| 11 | 38.234800 |
| 12 | 39.725669 |
| 13 | 41.008729 |
| 14 | 41.546766 |
| 15 | 43.814741 |
| 16 | 44.969343 |
| 17 | 45.787793 |
| 18 | 47.185149 |
| 19 | 47.394369 |
| 20 | 50.119177 |
| 21 | 58.528956 |
| 22 | 58.665499 |
| 23 | 58.744207 |
| 24 | 60.015172 |
| 25 | 61.159208 |
| 26 | 61.836861 |
| 27 | 64.025983 |
| 28 | 66.383295 |
| 29 | 67.772545 |
| 30 | 70.323378 |
| 31 | 72.601795 |
| 32 | 75.142853 |
| 33 | 81.278026 |
| 34 | 104.997635 |
| 35 | 107.536800 |
| 36 | 107.564509 |
| 37 | 112.748325 |
| 38 | 115.064815 |
| 39 | 116.749868 |
| 40 | 126.323208 |
| 41 | 126.476616 |
| 42 | 130.580217 |
| 43 | 142.322689 |
| 44 | 149.627343 |
| 45 | 166.032284 |
| 46 | 169.806391 |
| 47 | 175.272003 |
| 48 | 178.138345 |
| 49 | 178.688705 |
| 50 | 179.265440 |
| 51 | 202.122718 |
| 52 | 327.279150 |
| 53 | 485.229164 |
| 54 | 506.704867 |
| 55 | 510.604560 |
| 56 | 551.321268 |
| 57 | 579.006553 |
| 58 | 813.333809 |
| 59 | 989.103734 |

Fig 1 : Recorded frequency. Displayed in a dataframe

```
data=pd.read_csv("rec.csv")
#This is a function provided by Pandas to read a CSV (Comma Separated Values) file.
#It returns a DataFrame, which is a 2-dimensional labeled data structure with columns .

Range = data.freq
#assign all frequencies from rec.csv file to Range

y,sr = librosa.load("abnormal.wav")
#loading the abnormal.wav file

fre = y.max()*1000
#calculating frequency, that is from the all frequency, finding the max amplitude of the signal
# Load the first audio file and its sample rate using librosa
y, sr = librosa.load(audio_files[0])

# Create a plot of the audio time series data
pd.Series(y).plot(figsize=(10, 5), lw=1, title='Test 1 - wave plot')

# Add a grid to the plot (note: this line should call the function with parentheses)
plt.grid()

# Display the plot (note: this line should call the function with parentheses)
plt.show()

# Compute the Short-Time Fourier Transform (STFT) of the audio signal
D = librosa.stft(y)
```

```
# Convert the amplitude of the STFT to decibels
S_db = librosa.amplitude_to_db(np.abs(D), ref=np.max)

# Create a figure and axis for the plot
fig, ax = plt.subplots(figsize=(10, 5))

# Display the spectrogram with a logarithmic frequency axis
img = librosa.display.specshow(S_db, x_axis='time', y_axis='log', ax=ax)

# Set the title of the spectrogram plot
plt.title("spectrum")

# Display the spectrogram plot
plt.show()

print(" The given audiosfrequency is ",fre)
# printing the frequency of the uploaded 'abnormal.wav' file

if(fre>0 and fre<Range[30]):
  print("THE GIVEN SAMPLE IS CALM")
# if the fre is greater than 0 and less than 30 ,then the audio signal is Calm.
# here all the frequencies that we saved in rec.csv file is saved in Range, so comparing the value from the range of frequencies.

elif(fre>Range[30] and fre<Range[50]):
 print("THE GIVEN SAMPLE IS NORMAL")
# if the fre is greater than 30 and less than 50 ,then the audio signal is Normal.

elif(fre>Range[50]):
 print("THE GIVEN SAMPLE IS ABNORMAL")
# if the fre is greater than 50 then the audio signal is Abnormal.
```

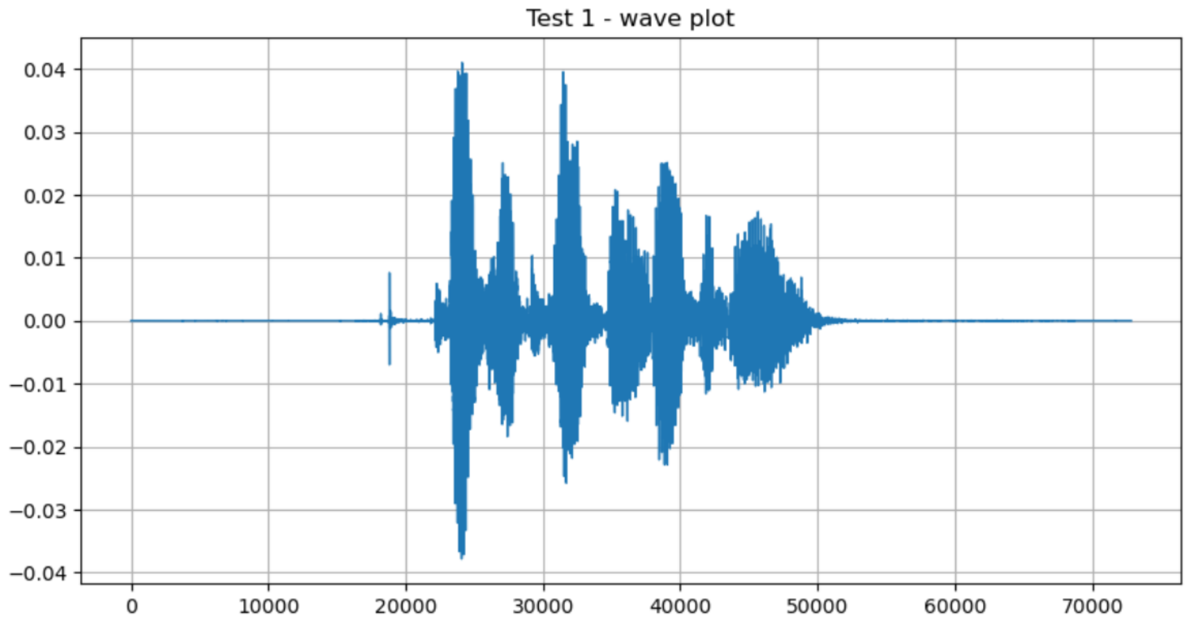


Fig 2 : The waveform of the signal(abnormal wave considered)

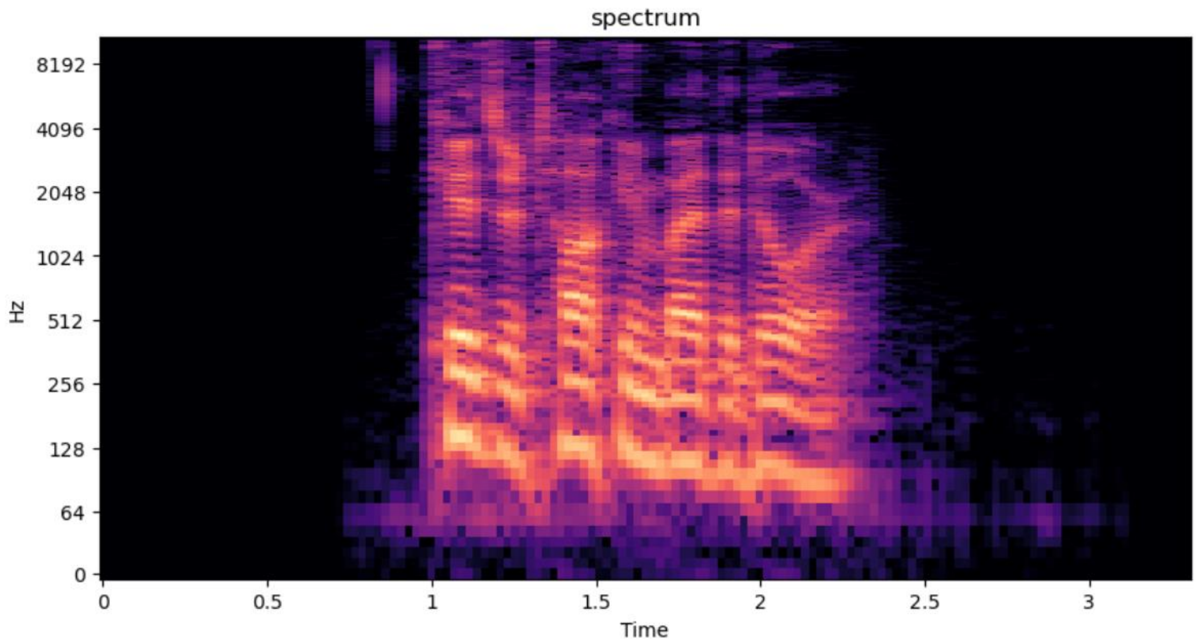


Fig 3 : The spectrum of the signal or waveform

```
 The given audiosfrequency is  241.27447605133057
THE GIVEN SAMPLE IS ABNORMAL
```

Fig 5 : The output showing the frequency and the type of audio signal based on the given conditions

## INFERENCE

### Feature Extraction:

The code uses `librosa.load` to get the maximum amplitude from audio files.

### Data Preparation:

The extracted amplitudes are stored, sorted, converted into a DataFrame, and saved as a CSV file.

### Labelling:

Audio samples are labelled as "Calm," "Normal," or "Abnormal" based on amplitude ranges, creating target variables for supervised learning.

### Classification:

The provided audio sample (`abnormal.wav`) is classified by comparing its maximum amplitude to the stored ranges, simulating how a supervised model makes predictions.

**TASK 1b:** Performance Analysis of Unsupervised learning – K means

```
#TASK 1(b) : PERFORMANCE ANALYSIS OF UNSUPERVISED LEARNING
# 24MEC0024
#DURGA SITALAKSHMI S
#PHONE NUMBER : 9731850288
#MEIC501P

import pandas as pd
# Importing the pandas library for data manipulation and analysis

import librosa
# Importing the librosa library for audio and music processing

import numpy as np
# Importing the numpy library for numerical operations on arrays

from sklearn.cluster import KMeans
# Importing the KMeans clustering algorithm from scikit-learn library

from sklearn.preprocessing import StandardScaler
# Importing StandardScaler for feature scaling from scikit-learn library

from matplotlib import pyplot as plt
# Importing pyplot module from matplotlib library for plotting

from glob import glob
# Importing glob module to find all pathnames matching a specified pattern

import warnings
# Importing warnings library to manage warnings
```

```python
warnings.filterwarnings("ignore")
# Ignores all warnings to prevent them from being displayed

audio_files =glob('Downloads\Audio_files/*.wav')
# Finds all .wav audio files in the specified directory using glob

mfccs = []
# Initializes an empty list to store MFCC (Mel-frequency cepstral coefficients) features

spectral_centroid =[]
# Initializes an empty list to store spectral centroid features

l=len(audio_files)
# Stores the number of audio files found in the directory

print(l)
```

```
<>:34: SyntaxWarning: invalid escape sequence '\A'
<>:34: SyntaxWarning: invalid escape sequence '\A'
C:\Users\sdurg\AppData\Local\Temp\ipykernel_19060\2430285894.py:34: SyntaxWarning: invalid escape sequence '\A'
  audio_files =glob('Downloads\Audio_files/*.wav')
6
```

```python
# Initialize an empty list for MFCCs and spectral centroids
mfccs = []
spectral_centroid = []

# Iterate over the range of audio files
for i in range(l):
    # Load the audio file and its sample rate
    animal, sr = librosa.load(audio_files[i])
    # Extract MFCC features with 20 coefficients
    mfccs_anim = librosa.feature.mfcc(y=animal, sr=sr, n_mfcc=20)
    # Extract spectral centroid features
    spectral_centroids_anims = librosa.feature.spectral_centroid(y=animal, sr=sr)
    # Append the maximum value of the 10th MFCC coefficient to the list
    mfccs.append(max(mfccs_anim[9]))
    # Append the maximum spectral centroid value to the list
    spectral_centroid.append(max(max(spectral_centroids_anims)))

# Create a DataFrame with MFCC and spectral centroid values
df = pd.DataFrame()
df['mfcc'] = mfccs
df['spectral'] = spectral_centroid

# Generate descriptive statistics for the DataFrame
stats_summary = df.describe()

# Standardize the MFCC and spectral centroid values
scaler = StandardScaler()
df[['mfcc_t', 'spectral_t']] = scaler.fit_transform(df[['mfcc', 'spectral']])
```

```python
# Apply KMeans clustering with 2 clusters
KM = KMeans(n_clusters=2)
y_predict = KM.fit_predict(df[['mfcc', 'spectral']])

# Add the cluster labels to the DataFrame
df['cluster'] = y_predict

# Print the DataFrame with the cluster labels
print(df)

# Plot the standardized MFCC vs. spectral centroid with cluster coloring
plt.scatter(df['mfcc_t'], df['spectral_t'], c=df['cluster'])
#This line plots the standardized MFCC values (df['mfcc_t']) against the standardized spectral centroid values (df['spectral_t']).
#The c=df['cluster'] argument uses the cluster labels generated by the KMeans algorithm to color the points.
#Each unique value in df['cluster'] will be assigned a different color.
plt.grid()
plt.xlabel('MFCC')
plt.ylabel('SPECTRAL_CENTROID')
plt.title("CLUSTER PLOT")
plt.show()

# Generate descriptive statistics again (if needed for later use)
stats_summary = df.describe()
```

```python
# Create a dictionary to store the summary statistics
summary_stats = {
    'mean': stats_summary.loc['mean', ['mfcc', 'spectral', 'mfcc_t', 'spectral_t', 'cluster']],
    'std': stats_summary.loc['std', ['mfcc', 'spectral', 'mfcc_t', 'spectral_t', 'cluster']],
    'min': stats_summary.loc['min', ['mfcc', 'spectral', 'mfcc_t', 'spectral_t', 'cluster']],
    '25%': stats_summary.loc['25%', ['mfcc', 'spectral', 'mfcc_t', 'spectral_t', 'cluster']],
    '50%': stats_summary.loc['50%', ['mfcc', 'spectral', 'mfcc_t', 'spectral_t', 'cluster']],
    '75%': stats_summary.loc['75%', ['mfcc', 'spectral', 'mfcc_t', 'spectral_t', 'cluster']],
    'max': stats_summary.loc['max', ['mfcc', 'spectral', 'mfcc_t', 'spectral_t', 'cluster']]
}

# Convert the dictionary to a DataFrame
summary_df = pd.DataFrame(summary_stats)

# Print the summary DataFrame
print(summary_df)
```

| | mfcc | spectral | mfcc_t | spectral_t | cluster |
|---|---|---|---|---|---|
| 0 | 68.688110 | 2410.670365 | 2.071106 | -0.496861 | 1 |
| 1 | 33.581944 | 2185.434301 | 0.134933 | -0.907513 | 1 |
| 2 | 31.278986 | 1973.660512 | 0.007921 | -1.293619 | 1 |
| 3 | 19.093605 | 3562.220285 | -0.664126 | 1.602649 | 0 |
| 4 | 19.957207 | 3098.451969 | -0.616497 | 0.757105 | 0 |
| 5 | 14.212361 | 2868.710377 | -0.933336 | 0.338239 | 0 |

Fig 6 : The dataframe of the mfcc, mfcc_t , spectrum , spectrum_t and the cluster category

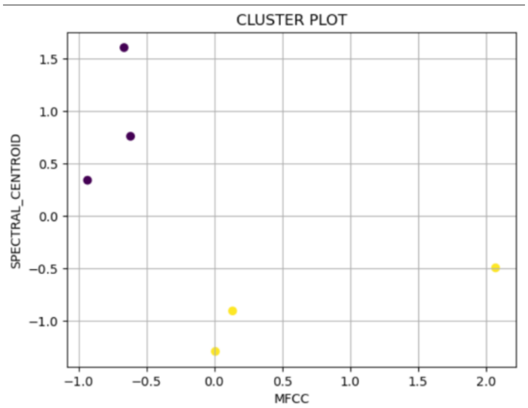


Fig 7 : The cluster plot

| | mean | std | min | 25% | 50% |
|---|---|---|---|---|---|
| mfcc | 3.113537e+01 | 19.862318 | 14.212361 | 19.309505 | 25.618096 |
| spectral | 2.683191e+03 | 600.835141 | 1973.660512 | 2241.743317 | 2639.690371 |
| mfcc_t | -3.700743e-17 | 1.095445 | -0.933336 | -0.652219 | -0.304288 |
| spectral_t | -4.163336e-16 | 1.095445 | -1.293619 | -0.804850 | -0.079311 |
| cluster | 5.000000e-01 | 0.547723 | 0.000000 | 0.000000 | 0.500000 |

| | 75% | max |
|---|---|---|
| mfcc | 33.006204 | 68.688110 |
| spectral | 3041.016571 | 3562.220285 |
| mfcc_t | 0.103180 | 2.071106 |
| spectral_t | 0.652389 | 1.602649 |
| cluster | 1.000000 | 1.000000 |

Fig 8 : The mean, standard deviation of mfcc, spectral, Spectral_t and cluster

```
# Load the first audio file and its sample rate using librosa
y, sr = librosa.load(audio_files[0])

# Create a plot of the audio time series data
pd.Series(y).plot(figsize=(10, 5), lw=1, title='Test 1 - wave plot')

# Add a grid to the plot (note: this line should call the function with parentheses)
plt.grid()

# Display the plot (note: this line should call the function with parentheses)
plt.show()

# Compute the Short-Time Fourier Transform (STFT) of the audio signal
D = librosa.stft(y)

# Convert the amplitude of the STFT to decibels
S_db = librosa.amplitude_to_db(np.abs(D), ref=np.max)

# Create a figure and axis for the plot
fig, ax = plt.subplots(figsize=(10, 5))

# Display the spectrogram with a logarithmic frequency axis
img = librosa.display.specshow(S_db, x_axis='time', y_axis='log', ax=ax)

# Set the title of the spectrogram plot
plt.title("spectrum")

# Display the spectrogram plot
plt.show()
```

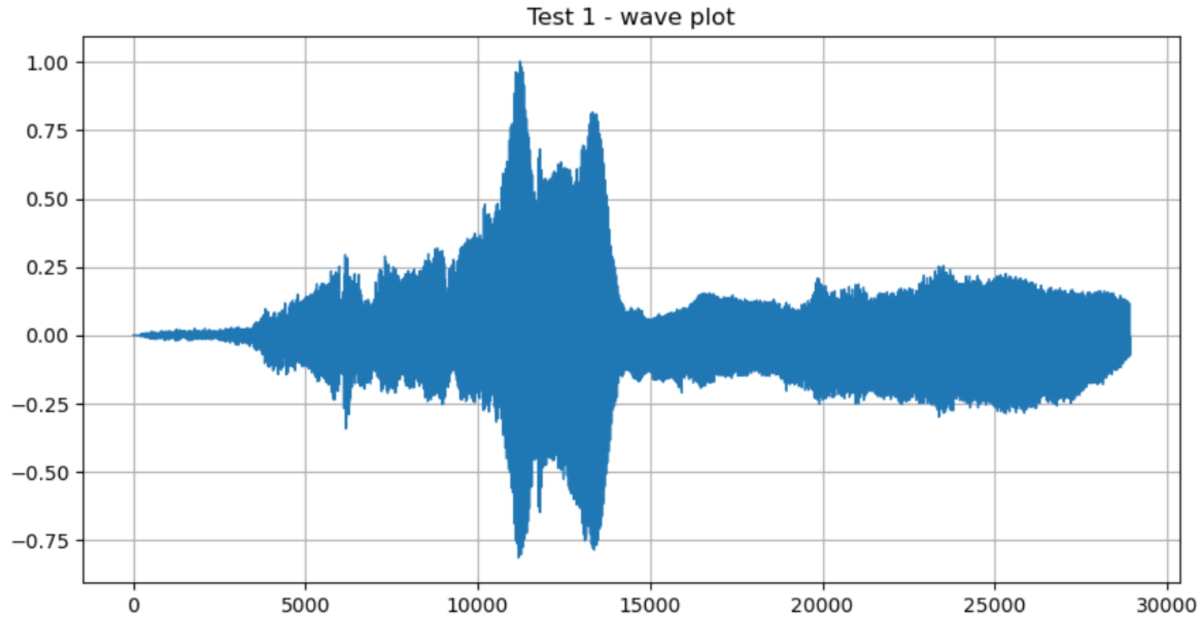


Fig 9 : Wave Plot

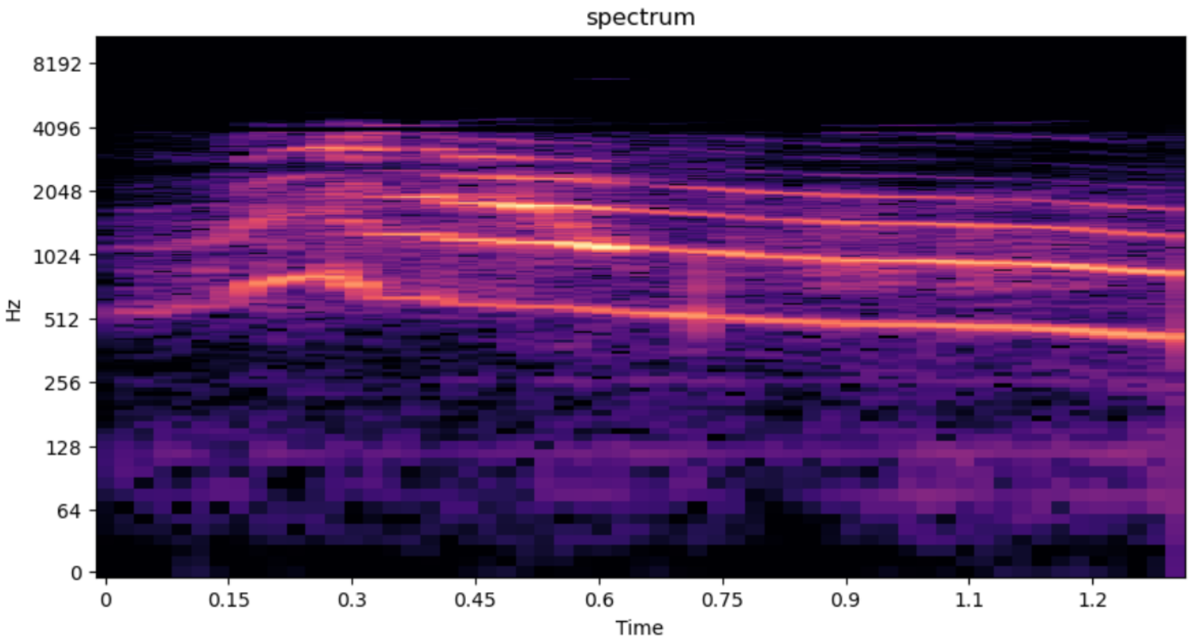


Fig 10 : Spectrum Plot

```python
# Compute MFCC (Mel-frequency cepstral coefficients) from the audio signal
mfc = librosa.feature.mfcc(y=y, sr=sr)

# Create a new figure for the plot with specified figure size
plt.figure(figsize=(10, 4))

# Display the MFCC spectrogram with time on the x-axis
librosa.display.specshow(mfc, x_axis='time')

# Add a colorbar to the plot with formatting for the decibel scale
plt.colorbar(format='%+2.0f dB')

# Set the title of the plot to 'MFCCs'
plt.title('MFCCs')

# Display the plot
plt.show()
```

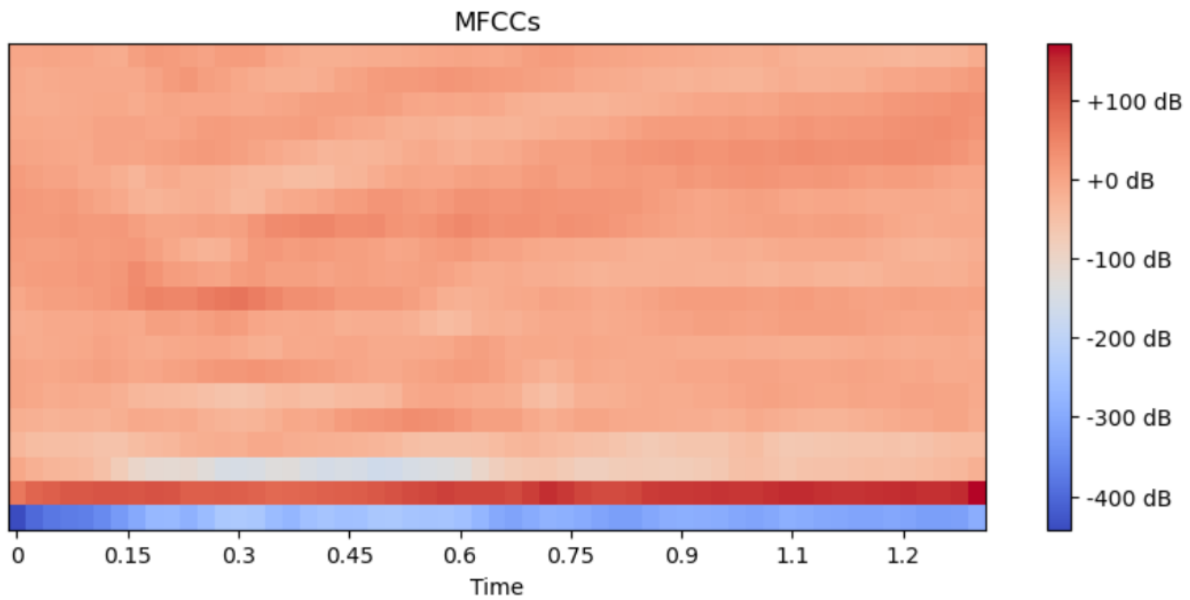


Fig 11 : MFCC Spectrum

**INFERENCE**

**Feature Extraction:**

The code uses librosa to extract MFCC and spectral centroid features from audio files. MFCCs capture essential spectral characteristics, while spectral centroids indicate the spectrum's "center of mass."

**Data Preparation:**

Extracted features are stored in lists, converted to a DataFrame, and standardized using StandardScaler, ensuring comparable feature scales for better clustering performance.

**Clustering:**

KMeans clustering groups the standardized features into two clusters, uncovering inherent patterns in the audio data without predefined labels.

**Visualization:**

The clustered data is visualized with a scatter plot of MFCC and spectral centroid features, illustrating cluster separation and groupings.

**Spectrogram and MFCC Visualization:**

The code plots the waveform, spectrogram (via STFT), and MFCCs of a sample audio file, providing detailed insights into the audio signal's structure for further analysis and cluster interpretation.

**Task 1c** : Supervised Learning - Analysing Support Vector Machine working

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import shutil
import matplotlib.pyplot as plt
import numpy as np
import librosa
import librosa.display
import IPython.display as ipd


# Output directory to clear
output_dir = "op"

# Clear the contents of the output directory
shutil.rmtree(output_dir, ignore_errors=True)
os.makedirs(output_dir, exist_ok=True)

print(f"Contents of {output_dir} cleared.")

import librosa
import soundfile as sf

# Path to the dataset
dataset_path = "Downloads/16000_pcm_speeches"
```

```python
# Output directory to save the combined files
output_dir = "op"

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# List of speaker folders
speaker_folders = [
    "Benjamin_Netanyau",
    "Jens_Stoltenberg",
    "Julia_Gillard",
    "Magaret_Tarcher",
    "Nelson_Mandela"
]
# Number of files to combine for each speaker
num_files_to_combine = 120

# Iterate over each speaker's folder
for speaker_folder in speaker_folders:
    speaker_folder_path = os.path.join(dataset_path, speaker_folder)

    # List the first num_files_to_combine WAV files in the speaker's folder
    wav_files = [f"{i}.wav" for i in range(num_files_to_combine)]
```

```python
    # Combine all WAV files into a single long file
    combined_audio = []
    for wav_file in wav_files:
        wav_file_path = os.path.join(speaker_folder_path, wav_file)
        audio, sr = librosa.load(wav_file_path, sr=None)
        combined_audio.extend(audio)

    # Save the combined audio file
    output_file_path = os.path.join(output_dir, f"{speaker_folder}_combined.wav")
    sf.write(output_file_path, combined_audio, sr)

print("Combination complete. Combined files saved in:", output_dir)
```

```
Contents of op cleared.
Combination complete. Combined files saved in: op
```

Fig 12 :Output of the first part of the code to combine the files

```python
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

# Set the parent directory for speaker folders
parent_dir = "Downloads/16000_pcm_speeches"

# List of speaker folders
speaker_folders = [
    "Benjamin_Netanyau",
    "Jens_Stoltenberg",
    "Julia_Gillard",
    "Magaret_Tarcher",
    "Nelson_Mandela"
]
```

```python
def extract_features(parent_dir, speaker_folders):
    features = []
    labels = []

    for i, speaker_folder in enumerate(speaker_folders):
        speaker_folder_path = os.path.join(parent_dir, speaker_folder)

        for filename in os.listdir(speaker_folder_path):
            if filename.endswith(".wav"):
                file_path = os.path.join(speaker_folder_path, filename)
                audio, sr = librosa.load(file_path, sr=None, duration=1)
                mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)

                # Normalize MFCC features
                mfccs = StandardScaler().fit_transform(mfccs)
                features.append(mfccs.T)
                labels.append(i)

    return np.array(features), np.array(labels)
```

```python
 # Extract features and labels
X, y = extract_features(parent_dir, speaker_folders)

# Print the first few features
for feature in X[:1]:
    print(feature)

from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import LabelEncoder

# Encode labels with explicit classes
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
label_encoder.classes_ = np.array(speaker_folders)

# Split the data into training, validation, and test sets
#X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
#X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Print the shapes of training and validation data
print("Training Data Shape:", X_train.shape)
print("Test Data Shape:", X_test.shape)
#print("Validation Data Shape:", X_val.shape)
```

```
 [[-3.46410155e+00  2.88717210e-01  2.88714826e-01  2.88710833e-01
    2.88705289e-01  2.88698137e-01  2.88689405e-01  2.88679123e-01
    2.88667232e-01  2.88653851e-01  2.88638890e-01  2.88622409e-01
    2.88604409e-01]
  [-3.46410155e+00  2.88697690e-01  2.88696378e-01  2.88694263e-01
    2.88691312e-01  2.88687497e-01  2.88682818e-01  2.88677305e-01
    2.88670957e-01  2.88663775e-01  2.88655698e-01  2.88646817e-01
    2.88637102e-01]
  [-3.46410179e+00  2.88694620e-01  2.88693517e-01  2.88691700e-01
    2.88689107e-01  2.88685828e-01  2.88681775e-01  2.88677037e-01
    2.88671494e-01  2.88665295e-01  2.88658351e-01  2.88650692e-01
    2.88642257e-01]
  [-3.41496515e+00  4.01689023e-01 -5.24658822e-02  6.99484289e-01
    2.78740406e-01  2.13533744e-01  3.30969393e-01  3.79182965e-01
    6.90628961e-02  2.89402425e-01  2.54391849e-01  3.14149350e-01
    2.36824691e-01]
  [-3.32731915e+00  5.34021258e-01 -3.14830184e-01  9.45635557e-01
    3.05575788e-01  2.24249974e-01  2.92698741e-01  3.76336753e-01
```

Fig 13 :Output of the features

```
Training Data Shape: (6000, 32, 13)
Test Data Shape: (1501, 32, 13)
```

Fig 14 :The output of the shape of the training and testing set

```python
from sklearn.mixture import GaussianMixture
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Gaussian Mixture Model (GMM)
#gmm = GaussianMixture(n_components=10, covariance_type='diag')
#gmm.fit(X_train, y_train)

# Extract features using GMM
#X_train_features = gmm.predict_proba(X_train)
#X_test_features = gmm.predict_proba(X_test)
```


```python
from sklearn.mixture import GaussianMixture
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Gaussian Mixture Model (GMM)
#gmm = GaussianMixture(n_components=10, covariance_type='diag')
#gmm.fit(X_train, y_train)

# Extract features using GMM
#X_train_features = gmm.predict_proba(X_train)
#X_test_features = gmm.predict_proba(X_test)
```

```python
import numpy as np

# Flatten the input data
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Initialize and train the SVM classifier
svm_classifier = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_classifier.fit(X_train_flat, y_train)

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Make predictions on the test set
y_pred = svm_classifier.predict(X_test_flat)
```

```python
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)

# Compute confusion matrix
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion_mat)

# Generate classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)
```

```python
import seaborn as sns

# Plot the confusion matrix
plt.figure(figsize=(7, 5))
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap="Blues", xticklabels=speaker_folders, yticklabels=speaker_folders)

# Rotate x-axis labels by 45 degrees
plt.xticks(rotation=45, ha="right")

plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Test Accuracy: 0.9713524317121919
Confusion Matrix:
[[304   4   0   2   0]
 [ 24 283   2   1   0]
 [  2   0 280   1   0]
 [  3   3   1 276   0]
 [  0   0   0   0 315]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.98      0.95       310
           1       0.98      0.91      0.94       310
           2       0.99      0.99      0.99       283
           3       0.99      0.98      0.98       283
           4       1.00      1.00      1.00       315

    accuracy                           0.97      1501
   macro avg       0.97      0.97      0.97      1501
weighted avg       0.97      0.97      0.97      1501
```

Fig 15 : Output of the test accuracy, confusion matrix, classification report and average

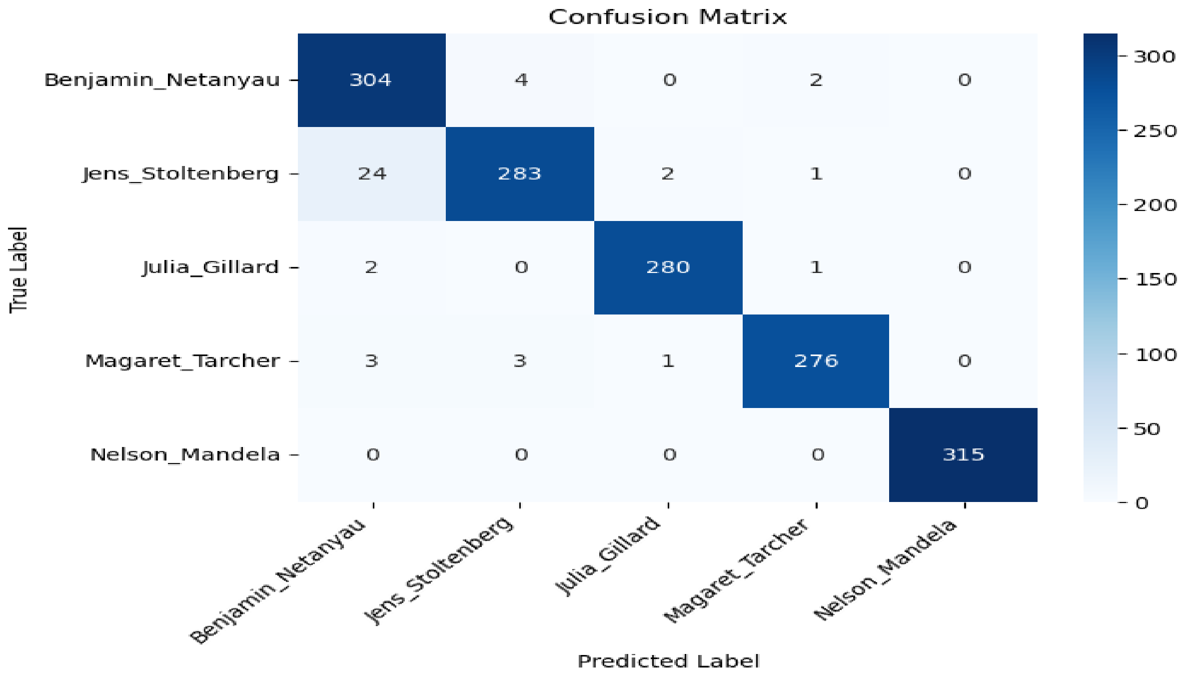


Fig 16 :Output showing the graph of the confusion matrix – heat map

INFERENCE

**Feature Extraction:**

This code uses the librosa library to extract features from audio files, specifically focusing on MFCC (Mel-Frequency Cepstral Coefficients) features. These MFCCs are crucial for capturing the unique characteristics of the audio signals.

**Data Preparation:**

The extracted MFCC features are collected in lists and then converted into a NumPy array. This structured data is split into training and testing sets using train_test_split, ensuring that the model is trained and evaluated on different subsets of data.

**Label Encoding:**

The labels representing different speakers are transformed into numerical format using Label Encoder. This conversion is necessary for the machine learning model to process and understand the categorical data.

**Data Flattening:**

The input data is reshaped into a 2D array where each row represents a sample with its corresponding features. This format is required by the SVM classifier.

**SVM Classification:**

An SVM (Support Vector Machine) classifier is initialized and trained using the training data. The training process involves finding the optimal hyperplane that separates the classes (different speakers) in the feature space.

**Prediction and Evaluation:**

The trained SVM model makes predictions on the test set. The model's performance is evaluated using various metrics:

- **Accuracy Score:** Measures the proportion of correctly classified samples.
- **Classification Report:** Provides detailed metrics such as precision, recall, and F1-score for each class.
- **Confusion Matrix:** Shows the number of correct and incorrect predictions for each class, offering deeper insights into the model's performance.

**Visualization:**

A heatmap of the confusion matrix is created to visualize the classifier's performance. The heatmap illustrates the true labels on the y-axis and the predicted labels on the x-axis, with color intensity indicating the number of samples in each category.