**🎧 Purpose of the Project:**

The goal of this project is to create an **audio classification model** that can identify different audio categories like **music**, **dialogue**, and **sound effects** based on the audio features extracted from files.

**1. Dataset (UrbanSound8K)**

The dataset used is **UrbanSound8K**, which contains **10,000+ urban sound clips**. These clips are labeled with different classes such as:

* Music
* Dialogue (speech)
* Sound effects (e.g., drilling, sirens)

For this project, we used only a subset of these categories to build the classifier:

* **music**
* **dialogue (speech)**
* **sound effect (drilling)**

**2. Model Type**

I am using a **Random Forest Classifier** — an ensemble learning method that creates a large number of decision trees to predict the class.

* **Advantages**: Random forests are robust, can handle imbalanced datasets, and are effective for classification problems.
* **Model**: RandomForestClassifier with n\_estimators=200 — meaning 200 trees are created in the forest.

**📝 Explanation of Code**

**Part 1: app.py — The Streamlit Web Application**

This file runs the web interface, where users can upload audio files, and it predicts the class of the audio. Let’s go through it step by step:

**1. Import Libraries**

python

import streamlit as st

import librosa

import numpy as np

import joblib

import tempfile

• **streamlit**: For creating the interactive web interface.

• **librosa**: A library for analyzing audio signals, especially useful for feature extraction.

• **numpy**: For handling arrays and numerical operations.

• **joblib**: To load the trained model (audio\_model.pkl).

• **tempfile**: Used to temporarily store the uploaded audio file.

**2. Load the Model**

python

@st.cache\_resource

def load\_model():

return joblib.load(“audio\_model.pkl”)

* **@st.cache\_resource**: This decorator caches the model so it isn’t loaded every time the app is used.
* **joblib.load()**: Loads the saved trained model (audio\_model.pkl) from the disk.

**3. Extract Features from Audio**

python

def extract\_features(file\_path):

audio, sample\_rate = librosa.load(file\_path, sr=None)

mfccs = librosa.feature.mfcc(y=audio, sr=sample\_rate, n\_mfcc=40)

mfccs\_mean = np.mean(mfccs, axis=1)

mfccs\_std = np.std(mfccs, axis=1)

return np.hstack([mfccs\_mean, mfccs\_std])

* **librosa.load()**: Loads the audio file and returns the waveform (audio) and sample rate (sample\_rate).
* **librosa.feature.mfcc()**: Extracts **MFCCs (Mel-frequency cepstral coefficients)**, which represent the power spectrum of the audio signal. These are commonly used features for audio classification tasks.
* **np.mean()** and **np.std()**: Compute the **mean** and **standard deviation** of the MFCCs, which help represent the overall characteristics of the audio.
* **np.hstack()**: Concatenates the mean and standard deviation of MFCCs into one feature vector.

**4. File Uploader (Streamlit)**

python

uploaded\_file = st.file\_uploader(“Upload an audio file (.wav or .mp3)”, type=[“wav”, “mp3”])

* Allows users to upload audio files (.wav or .mp3).

**5. Prediction and Display**

python

if uploaded\_file is not None:

st.audio(uploaded\_file, format=‘audio/wav’)

with st.spinner(“🔍 Extracting features and making prediction...”):

try:

with tempfile.NamedTemporaryFile(delete=False) as tmp\_file:

tmp\_file.write(uploaded\_file.read())

tmp\_path = tmp\_file.name

features = extract\_features(tmp\_path)

probs = model.predict\_proba([features])[0]

classes = model.classes\_

top\_index = np.argmax(probs)

top\_class = classes[top\_index]

confidence = probs[top\_index]

st.success(f”🎯 Predicted Class: \*\*{top\_class.capitalize()}\*\* ({confidence \* 100:.2f}% confidence)")

st.markdown(“### 🔍 Class Probabilities:”)

for cls, prob in zip(classes, probs):

st.markdown(f”- \*\*{cls.capitalize()}\*\*: {prob \* 100:.2f}%")

except Exception as e:

st.error(f”❌ Oops! Something went wrong during prediction.\n\nError: {e}")

* **st.audio()**: Displays the uploaded audio file in the app so users can listen to it.
* **st.spinner()**: Shows a loading spinner while the features are being extracted and the prediction is made.
* **extract\_features()**: This function extracts the necessary features from the uploaded audio.
* **model.predict\_proba()**: This predicts the probabilities for each class (music, dialogue, sound effect).
* **np.argmax()**: Finds the class with the highest probability.
* **st.success()**: Displays the top prediction along with the confidence percentage.
* **st.markdown()**: Displays the probability breakdown for all classes.

**Part 2: train\_audio\_classifier.py — The Model Training Script**

This file is responsible for **training the model** using the **UrbanSound8K** dataset. Here’s how it works:

**1. Import Libraries**

python

import os

import librosa

import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report, ConfusionMatrixDisplay

from sklearn.utils import resample

import matplotlib.pyplot as plt

import joblib

import warnings

warnings.filterwarnings(“ignore”)

* **librosa**: Used to load and process the audio files.
* **pandas**: For handling the dataset (CSV format).
* **RandomForestClassifier**: The machine learning model used for classification.
* **train\_test\_split**: To split the data into training and test sets.
* **accuracy\_score** and **classification\_report**: For evaluating the model’s performance.

**2. Extract Features**

python

def extract\_features(file\_path):

try:

y, sr = librosa.load(file\_path, sr=None)

if len(y) < sr:

raise ValueError(“Audio too short”)

mfccs = librosa.feature.mfcc(y=y, sr=sr, n\_mfcc=40)

mfccs\_mean = np.mean(mfccs, axis=1)

mfccs\_std = np.std(mfccs, axis=1)

return np.hstack([mfccs\_mean, mfccs\_std])

except Exception as e:

print(f”Skipped {file\_path}: {e}")

return None

* This is similar to the feature extraction function in app.py. It processes the audio files and extracts the MFCCs.

**3. Load and Process the Data**

python

def load\_data():

df = pd.read\_csv(CSV\_PATH)

df = df[df[‘class’].isin([‘speech’, ‘music’, ‘drilling’])]

...

return np.array(X), np.array(y)

* **pd.read\_csv()**: Reads the CSV file that contains metadata about the dataset.
* **df[df[‘class’].isin()]**: Filters the data to only keep classes for speech, music, and sound effects (drilling).
* The features are extracted for each audio file, and the corresponding labels are added to X and y.

**4. Train the Model**

python

model = RandomForestClassifier(n\_estimators=200, random\_state=42)

model.fit(X\_train, y\_train)

* **RandomForestClassifier**: Trains the model using the extracted features and labels.
* **n\_estimators=200**: Specifies the number of trees in the forest.

**5. Evaluate the Model**

python

y\_pred = model.predict(X\_test)

print(f”\n✅ Accuracy: {accuracy\_score(y\_test, y\_pred):.2f}")

print(“\nClassification Report:”)

print(classification\_report(y\_test, y\_pred))

* Evaluates the model using the test set and prints the accuracy and classification report.

**6. Save the Model**

python

joblib.dump(model, “audio\_model.pkl”, compress=3)

* Saves the trained model to a file (audio\_model.pkl), which is later loaded by the Streamlit app.

**🔑 Key Concepts**

**Feature Engineering**

The model is trained on **MFCCs**, which are a standard feature in audio classification. These features capture the timbre of the audio signal, which is useful for distinguishing between different sound classes.

**Random Forest**

A random forest is an ensemble of decision trees. Each tree makes an independent prediction, and the forest averages the results to make a final prediction. This helps reduce overfitting and increase robustness.

**📊 Model Performance**

The model achieves **over 90% accuracy** with balanced classes, as seen from the output during training. It uses metrics like **accuracy** and **F1 score** for evaluation, and a **confusion matrix** to understand the classification performance better.

**🚀 Conclusion**

I’ve built a **real-time audio classification app** using:

* **Feature extraction** with **MFCCs** using librosa
* **Random Forests** for classification
* **Streamlit** for a real-time, interactive web interface