!pip install speechrecognition textblob scikit-learn pandas joblib

!pip install pyttsx3  # For text-to-speech conversion (optional for better interaction)

import os

import speech\_recognition as sr

from textblob import TextBlob

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

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

import pandas as pd

import joblib

from google.colab import files

# -------- TRAIN MODEL (with evaluation metrics) --------

def train\_model():

    print("🔧 Training model...")

    data = {

        'SpeechText': [

            "I am happy with the service",

            "I'm very angry and frustrated",

            "Everything is fine",

            "I’m disappointed with this",

            "It’s really good",

            "Worst experience ever"

            ],

        'Churned': [0, 1, 0, 1, 0, 1]

    }

    df = pd.DataFrame(data)

    def extract\_features(text):

        blob = TextBlob(text)

        sentiment = blob.sentiment.polarity

        emotion\_words = ['angry', 'frustrated', 'sad', 'happy', 'disappointed', 'satisfied']

        count = sum(word in text.lower() for word in emotion\_words)

        return [sentiment, count]

    df[['Sentiment', 'EmotionCount']] = df['SpeechText'].apply(lambda x: pd.Series(extract\_features(x)))

    X = df[['Sentiment', 'EmotionCount']]

    y = df['Churned']

    # Train-test split for evaluation

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

    model = RandomForestClassifier(random\_state=42)

    model.fit(X\_train, y\_train)

    # Predict on test set

    y\_pred = model.predict(X\_test)

    # Evaluation metrics

    accuracy = accuracy\_score(y\_test, y\_pred)

    precision = precision\_score(y\_test, y\_pred, zero\_division=0)

    recall = recall\_score(y\_test, y\_pred, zero\_division=0)

    f1 = f1\_score(y\_test, y\_pred, zero\_division=0)

    cm = confusion\_matrix(y\_test, y\_pred)

    print("\n📈 Evaluation Metrics:")

    print(f"✅ Accuracy: {accuracy:.2f}")

    print(f"✅ Precision: {precision:.2f}")

    print(f"✅ Recall: {recall:.2f}")

    print(f"✅ F1 Score: {f1:.2f}")

    print("✅ Confusion Matrix:\n", cm)

    # Save model

    joblib.dump(model, "churn\_model.pkl")

    print("\n💾 Model saved as churn\_model.pkl")

    return model

# -------- UPLOAD AUDIO FILE --------

def upload\_audio():

    print("⏬ Upload an audio file (in .wav format) for speech-to-text conversion.")

    uploaded = files.upload()

    for audio\_file in uploaded.keys():

        return audio\_file

# -------- CONVERT AUDIO TO TEXT --------

def recognize\_speech\_from\_file(audio\_file):

    recognizer = sr.Recognizer()

    with sr.AudioFile(audio\_file) as source:

        print("🎧 Converting audio to text...")

        audio = recognizer.record(source)

    try:

        text = recognizer.recognize\_google(audio)

        print("📝 Transcribed Text:", text)

        return text

    except:

        print("❌ Could not understand the audio.")

        return ""

# -------- EXTRACT EMOTION FEATURES --------

def extract\_emotion\_features(text):

    blob = TextBlob(text)

    sentiment = blob.sentiment.polarity

    emotion\_words = ['angry', 'frustrated', 'sad', 'happy', 'disappointed', 'satisfied']

    count = sum(word in text.lower() for word in emotion\_words)

    return [sentiment, count]

# -------- MAIN --------

def main():

    model = train\_model()  # Always retrains and prints metrics

    audio\_file = upload\_audio()  # Upload your audio file

    if not audio\_file:

        print("❌ No audio file uploaded.")

        return

    text = recognize\_speech\_from\_file(audio\_file)  # Convert audio to text

    if not text:

        return

    features = extract\_emotion\_features(text)

    print("📊 Features: Sentiment =", features[0], ", Emotion Count =", features[1])

    prediction = model.predict([features])[0]

    print("🔮 Prediction:", "CHURN" if prediction == 1 else "STAY")

if \_\_name\_\_ == "\_\_main\_\_":

    main()