Proyecto

June 4, 2025

1 Analizador de Emociones por Voz mediante Inteligencia Artificial

1.1 1. Instalación e Importación de Librerías

```
# Instalación de librerías necesarias
!pip install librosa soundfile numpy pandas scikit-learn tensorflow matplotlib seaborn

import librosa
import soundfile as sf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Conv1D, MaxPooling1D, Flatten
import warnings
warnings.filterwarnings('ignore')
```

1.2 2. Funciones para Carga y Preprocesamiento de Audio

```
def load_audio_file(file_path, sample_rate=22050):
    """
    Carga un archivo de audio y lo convierte al sample rate especificado
    """
    try:
        audio, sr = librosa.load(file_path, sr=sample_rate)
        return audio, sr
    except Exception as e:
        print(f"Error cargando archivo {file_path}: {e}")
        return None, None

def preprocess_audio(audio, sr=22050, duration=3.0):
    """
```

```
    Preprocesa el audio: normalización y ajuste de duración
    """
    # Normalizar audio
    audio = librosa.util.normalize(audio)

    # Ajustar duración (padding o truncado)
    target_length = int(sr * duration)
    if len(audio) > target_length:
        audio = audio[:target_length]
    else:
        audio = np.pad(audio, (0, target_length - len(audio)), mode='constant')

    return audio
```

1.3 3. Extracción de Características de Audio

```
def extract_features(audio, sr=22050):
    """
    Extrae características relevantes del audio para análisis de emociones
    """
    features = {}

    # 1. Características espectrales
    # MFCC (Mel-frequency cepstral coefficients)
    mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
    features['mfcc_mean'] = np.mean(mfccs, axis=1)
    features['mfcc_std'] = np.std(mfccs, axis=1)

    # Espectrograma Mel
    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sr)
    features['mel_mean'] = np.mean(mel_spec, axis=1)
    features['mel_std'] = np.std(mel_spec, axis=1)

    # 2. Características de tono
    # Frecuencia fundamental (F0)
    pitches, magnitudes = librosa.piptrack(y=audio, sr=sr)
    pitch_values = []
    for t in range(pitches.shape[1]):
        index = magnitudes[:, t].argmax()
        pitch = pitches[index, t]
        if pitch > 0:
            pitch_values.append(pitch)

    if pitch_values:
        features['pitch_mean'] = np.mean(pitch_values)
        features['pitch_std'] = np.std(pitch_values)
        features['pitch_min'] = np.min(pitch_values)
        features['pitch_max'] = np.max(pitch_values)
```

```
    else:
        features['pitch_mean'] = features['pitch_std'] = 0
        features['pitch_min'] = features['pitch_max'] = 0

    # 3. Características de energía
    # RMS (Root Mean Square) - energía
    rms = librosa.feature.rms(y=audio)
    features['rms_mean'] = np.mean(rms)
    features['rms_std'] = np.std(rms)

    # Zero Crossing Rate
    zcr = librosa.feature.zero_crossing_rate(audio)
    features['zcr_mean'] = np.mean(zcr)
    features['zcr_std'] = np.std(zcr)

    # 4. Características temporales
    # Centroide espectral
    spectral_centroids = librosa.feature.spectral_centroid(y=audio, sr=sr)
    features['spectral_centroid_mean'] = np.mean(spectral_centroids)
    features['spectral_centroid_std'] = np.std(spectral_centroids)

    # Rolloff espectral
    spectral_rolloff = librosa.feature.spectral_rolloff(y=audio, sr=sr)
    features['spectral_rolloff_mean'] = np.mean(spectral_rolloff)
    features['spectral_rolloff_std'] = np.std(spectral_rolloff)

    # Convertir a array plano
    feature_vector = []
    for key in sorted(features.keys()):
        if isinstance(features[key], np.ndarray):
            feature_vector.extend(features[key])
        else:
            feature_vector.append(features[key])

    return np.array(feature_vector)
```

1.4 4. Preparación del Dataset

```
def prepare_dataset(audio_directory, emotions_mapping):
    """
    Prepara el dataset extrayendo características de todos los archivos de audio

    audio_directory: directorio con archivos de audio organizados por emoción
    emotions_mapping: diccionario que mapea nombres de carpetas a emociones
    """
    features_list = []
    emotions_list = []
```

```
    import os

    for emotion_folder in os.listdir(audio_directory):
        if emotion_folder in emotions_mapping:
            emotion_path = os.path.join(audio_directory, emotion_folder)
            emotion_label = emotions_mapping[emotion_folder]

            for audio_file in os.listdir(emotion_path):
                if audio_file.endswith(('.wav', '.mp3', '.m4a')):
                    file_path = os.path.join(emotion_path, audio_file)

                    # Cargar y preprocesar audio
                    audio, sr = load_audio_file(file_path)
                    if audio is not None:
                        audio = preprocess_audio(audio, sr)

                        # Extraer características
                        features = extract_features(audio, sr)

                        features_list.append(features)
                        emotions_list.append(emotion_label)

    return np.array(features_list), np.array(emotions_list)

# Ejemplo de mapeo de emociones
emotions_mapping = {
    'happy': 'Felicidad',
    'sad': 'Tristeza',
    'angry': 'Enojo',
    'fear': 'Miedo',
    'neutral': 'Neutral',
    'surprise': 'Sorpresa',
    'disgust': 'Disgusto'
}
```

1.5 5. Exploración y Visualización de Datos

```
def visualize_audio_features(features_df, emotions):
    """
    Visualiza las características extraídas del audio
    """
    plt.figure(figsize=(15, 10))

    # Distribución de emociones
    plt.subplot(2, 3, 1)
    emotion_counts = pd.Series(emotions).value_counts()
    plt.pie(emotion_counts.values, labels=emotion_counts.index, autopct='%1.1f%%')
    plt.title('Distribución de Emociones')
```

```
# Características MFCC promedio por emoción
plt.subplot(2, 3, 2)
mfcc_cols = [col for col in features_df.columns if 'mfcc_mean' in col]
mfcc_data = features_df[mfcc_cols].values

for i, emotion in enumerate(np.unique(emotions)):
    emotion_mask = emotions == emotion
    plt.plot(np.mean(mfcc_data[emotion_mask], axis=0), label=emotion)

plt.xlabel('Coeficientes MFCC')
plt.ylabel('Valor Promedio')
plt.title('MFCC Promedio por Emoción')
plt.legend()

# Pitch promedio por emoción
plt.subplot(2, 3, 3)
pitch_data = []
emotion_labels = []

for emotion in np.unique(emotions):
    emotion_mask = emotions == emotion
    pitch_values = features_df.loc[emotion_mask, 'pitch_mean'].values
    pitch_data.extend(pitch_values)
    emotion_labels.extend([emotion] * len(pitch_values))

sns.boxplot(x=emotion_labels, y=pitch_data)
plt.xticks(rotation=45)
plt.title('Distribución de Pitch por Emoción')

# Energía (RMS) por emoción
plt.subplot(2, 3, 4)
rms_data = []
emotion_labels = []

for emotion in np.unique(emotions):
    emotion_mask = emotions == emotion
    rms_values = features_df.loc[emotion_mask, 'rms_mean'].values
    rms_data.extend(rms_values)
    emotion_labels.extend([emotion] * len(rms_values))

sns.boxplot(x=emotion_labels, y=rms_data)
plt.xticks(rotation=45)
plt.title('Distribución de Energía (RMS) por Emoción')

# Zero Crossing Rate por emoción
plt.subplot(2, 3, 5)
zcr_data = []
```

```
    emotion_labels = []

    for emotion in np.unique(emotions):
        emotion_mask = emotions == emotion
        zcr_values = features_df.loc[emotion_mask, 'zcr_mean'].values
        zcr_data.extend(zcr_values)
        emotion_labels.extend([emotion] * len(zcr_values))

    sns.boxplot(x=emotion_labels, y=zcr_data)
    plt.xticks(rotation=45)
    plt.title('Zero Crossing Rate por Emoción')

    # Matriz de correlación
    plt.subplot(2, 3, 6)
    correlation_matrix = features_df.corr()
    sns.heatmap(correlation_matrix.iloc[:10, :10], annot=True, cmap='coolwarm')
    plt.title('Matriz de Correlación (Primeras 10 características)')

    plt.tight_layout()
    plt.show()
```

1.6 6. Modelos de Machine Learning

```
class EmotionClassifier:
    def __init__(self, model_type='neural_network'):
        self.model_type = model_type
        self.model = None
        self.scaler = StandardScaler()
        self.label_encoder = LabelEncoder()

    def create_neural_network(self, input_shape, num_classes):
        """
        Crea una red neuronal profunda para clasificación de emociones
        """
        model = Sequential([
            Dense(512, activation='relu', input_shape=(input_shape,)),
            Dropout(0.3),
            Dense(256, activation='relu'),
            Dropout(0.3),
            Dense(128, activation='relu'),
            Dropout(0.2),
            Dense(64, activation='relu'),
            Dropout(0.2),
            Dense(num_classes, activation='softmax')
        ])

        model.compile(
            optimizer='adam',
```

```
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']
        )

        return model

    def create_cnn_model(self, input_shape, num_classes):
        """
        Crea un modelo CNN para análisis de características temporales
        """
        model = Sequential([
            tf.keras.layers.Reshape((input_shape, 1), input_shape=(input_shape,)),
            Conv1D(64, 3, activation='relu'),
            MaxPooling1D(2),
            Conv1D(128, 3, activation='relu'),
            MaxPooling1D(2),
            Conv1D(256, 3, activation='relu'),
            MaxPooling1D(2),
            Flatten(),
            Dense(512, activation='relu'),
            Dropout(0.3),
            Dense(256, activation='relu'),
            Dropout(0.3),
            Dense(num_classes, activation='softmax')
        ])

        model.compile(
            optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']
        )

        return model

    def train(self, X_train, y_train, X_val, y_val, epochs=100, batch_size=32):
        """
        Entrena el modelo de clasificación de emociones
        """
        # Normalizar características
        X_train_scaled = self.scaler.fit_transform(X_train)
        X_val_scaled = self.scaler.transform(X_val)

        # Codificar etiquetas
        y_train_encoded = self.label_encoder.fit_transform(y_train)
        y_val_encoded = self.label_encoder.transform(y_val)

        # Crear modelo
        num_classes = len(np.unique(y_train))
```

```
        input_shape = X_train_scaled.shape[1]

        if self.model_type == 'neural_network':
            self.model = self.create_neural_network(input_shape, num_classes)
        elif self.model_type == 'cnn':
            self.model = self.create_cnn_model(input_shape, num_classes)

        # Callbacks para entrenamiento
        callbacks = [
            tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True),
            tf.keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=5)
        ]

        # Entrenar modelo
        history = self.model.fit(
            X_train_scaled, y_train_encoded,
            validation_data=(X_val_scaled, y_val_encoded),
            epochs=epochs,
            batch_size=batch_size,
            callbacks=callbacks,
            verbose=1
        )

        return history

    def predict(self, X_test):
        """
        Realiza predicciones sobre nuevos datos
        """
        X_test_scaled = self.scaler.transform(X_test)
        predictions = self.model.predict(X_test_scaled)
        predicted_classes = np.argmax(predictions, axis=1)
        predicted_emotions = self.label_encoder.inverse_transform(predicted_classes)

        return predicted_emotions, predictions

    def evaluate(self, X_test, y_test):
        """
        Evalúa el rendimiento del modelo
        """
        y_pred, probabilities = self.predict(X_test)

        # Métricas de evaluación
        accuracy = accuracy_score(y_test, y_pred)
        report = classification_report(y_test, y_pred)
        conf_matrix = confusion_matrix(y_test, y_pred)

        return accuracy, report, conf_matrix
```

1.7 7. Entrenamiento y Evaluación del Modelo

```
# Ejemplo de uso del clasificador
def train_emotion_classifier(features, emotions):
    """
    Entrena y evalúa el clasificador de emociones
    """
    # Dividir datos en entrenamiento, validación y prueba
    X_train, X_temp, y_train, y_temp = train_test_split(
        features, emotions, test_size=0.4, random_state=42, stratify=emotions
    )
    X_val, X_test, y_val, y_test = train_test_split(
        X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp
    )

    print(f"Datos de entrenamiento: {X_train.shape[0]} muestras")
    print(f"Datos de validación: {X_val.shape[0]} muestras")
    print(f"Datos de prueba: {X_test.shape[0]} muestras")

    # Crear y entrenar clasificador
    classifier = EmotionClassifier(model_type='neural_network')
    history = classifier.train(X_train, y_train, X_val, y_val, epochs=50)

    # Evaluar modelo
    accuracy, report, conf_matrix = classifier.evaluate(X_test, y_test)

    print(f"\nPrecisión del modelo: {accuracy:.4f}")
    print(f"\nReporte de clasificación:\n{report}")

    # Visualizar resultados
    plot_training_history(history)
    plot_confusion_matrix(conf_matrix, classifier.label_encoder.classes_)

    return classifier

def plot_training_history(history):
    """
    Visualiza la historia de entrenamiento
    """
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Entrenamiento')
    plt.plot(history.history['val_accuracy'], label='Validación')
    plt.title('Precisión del Modelo')
    plt.xlabel('Época')
    plt.ylabel('Precisión')
    plt.legend()
```

```
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Entrenamiento')
    plt.plot(history.history['val_loss'], label='Validación')
    plt.title('Pérdida del Modelo')
    plt.xlabel('Época')
    plt.ylabel('Pérdida')
    plt.legend()

    plt.tight_layout()
    plt.show()

def plot_confusion_matrix(conf_matrix, class_names):
    """
    Visualiza la matriz de confusión
    """
    plt.figure(figsize=(10, 8))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('Matriz de Confusión')
    plt.xlabel('Predicción')
    plt.ylabel('Valor Real')
    plt.show()
```

1.8 8. Predicción en Tiempo Real

```
def real_time_emotion_prediction(model_path, audio_file_path):
    """
    Realiza predicción de emoción en tiempo real para un archivo de audio
    """
    # Cargar modelo entrenado
    classifier = EmotionClassifier()
    # classifier.model = tf.keras.models.load_model(model_path)

    # Procesar archivo de audio
    audio, sr = load_audio_file(audio_file_path)
    if audio is None:
        return "Error al cargar el archivo de audio"

    # Preprocesar y extraer características
    audio_processed = preprocess_audio(audio, sr)
    features = extract_features(audio_processed, sr)

    # Realizar predicción
    features_reshaped = features.reshape(1, -1)
    emotion, probabilities = classifier.predict(features_reshaped)

    return emotion[0], probabilities[0]
```

```python
def visualize_audio_waveform(audio_file_path):
    """
    Visualiza la forma de onda y espectrograma de un archivo de audio
    """
    audio, sr = load_audio_file(audio_file_path)

    plt.figure(figsize=(15, 10))

    # Forma de onda
    plt.subplot(3, 1, 1)
    time = np.linspace(0, len(audio)/sr, len(audio))
    plt.plot(time, audio)
    plt.title('Forma de Onda del Audio')
    plt.xlabel('Tiempo (s)')
    plt.ylabel('Amplitud')

    # Espectrograma
    plt.subplot(3, 1, 2)
    D = librosa.amplitude_to_db(np.abs(librosa.stft(audio)), ref=np.max)
    librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='hz')
    plt.colorbar(format='%+2.0f dB')
    plt.title('Espectrograma')

    # MFCC
    plt.subplot(3, 1, 3)
    mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
    librosa.display.specshow(mfccs, sr=sr, x_axis='time')
    plt.colorbar()
    plt.title('Coeficientes MFCC')

    plt.tight_layout()
    plt.show()
```

1.9 9. Ejemplo de Uso Completo

```python
# Ejemplo de uso completo del sistema
def main_emotion_analysis():
    """
    Función principal que ejecuta todo el pipeline de análisis de emociones
    """
    print("=== Analizador de Emociones por Voz ===\n")

    # 1. Preparar dataset (reemplazar con tu directorio de datos)
    # audio_directory = "path/to/your/audio/dataset"
    # features, emotions = prepare_dataset(audio_directory, emotions_mapping)

    # Para demostración, crear datos sintéticos
```

```
    np.random.seed(42)
    num_samples = 1000
    num_features = 50
    features = np.random.randn(num_samples, num_features)
    emotions = np.random.choice(['Felicidad', 'Tristeza', 'Enojo', 'Miedo', 'Neutral'],
                                size=num_samples)

    print(f"Dataset cargado: {features.shape[0]} muestras con {features.shape[1]} característi

    # 2. Explorar datos
    features_df = pd.DataFrame(features, columns=[f'feature_{i}' for i in range(features.shape
    # visualize_audio_features(features_df, emotions)

    # 3. Entrenar modelo
    classifier = train_emotion_classifier(features, emotions)

    # 4. Ejemplo de predicción en tiempo real
    # audio_file = "path/to/test/audio.wav"
    # emotion, confidence = real_time_emotion_prediction(classifier, audio_file)
    # print(f"Emoción detectada: {emotion} (Confianza: {np.max(confidence):.2f})")

    print("\n=== Análisis Completado ===")

    return classifier

# Ejecutar análisis principal
if __name__ == "__main__":
    classifier = main_emotion_analysis()
```

1.10 10. Utilidades Adicionales

```
def save_model(classifier, model_path, scaler_path, encoder_path):
    """
    Guarda el modelo entrenado y los preprocessors
    """
    classifier.model.save(model_path)

    import joblib
    joblib.dump(classifier.scaler, scaler_path)
    joblib.dump(classifier.label_encoder, encoder_path)

    print(f"Modelo guardado en: {model_path}")
    print(f"Scaler guardado en: {scaler_path}")
    print(f"Encoder guardado en: {encoder_path}")

def load_trained_model(model_path, scaler_path, encoder_path):
    """
    Carga un modelo previamente entrenado
```

```
    """
    import joblib

    classifier = EmotionClassifier()
    classifier.model = tf.keras.models.load_model(model_path)
    classifier.scaler = joblib.load(scaler_path)
    classifier.label_encoder = joblib.load(encoder_path)

    return classifier

def batch_emotion_analysis(audio_directory, classifier):
    """
    Analiza múltiples archivos de audio en lote
    """
    results = []

    import os
    for audio_file in os.listdir(audio_directory):
        if audio_file.endswith(('.wav', '.mp3', '.m4a')):
            file_path = os.path.join(audio_directory, audio_file)

            try:
                emotion, confidence = real_time_emotion_prediction(classifier, file_path)
                results.append({
                    'archivo': audio_file,
                    'emocion': emotion,
                    'confianza': np.max(confidence)
                })
            except Exception as e:
                print(f"Error procesando {audio_file}: {e}")

    return pd.DataFrame(results)
```

Este notebook proporciona un sistema completo para análisis de emociones por voz usando inteligencia artificial, incluyendo:

- Preprocesamiento de audio
- Extracción de características acústicas
- Modelos de deep learning
- Evaluación y visualización
- Predicción en tiempo real
- Utilidades para guardar/cargar modelos

Para usar este código, necesitarás un dataset de audio etiquetado con emociones organizizado en carpetas por cada emoción.