Audio Classification with Random Forest

May 5, 2025

Introduction

This document demonstrates the steps involved in training a Random Forest model for audio classification. The task involves preprocessing audio data, extracting relevant features, training a model, and evaluating the model using various metrics, including the Root Mean Squared Error (RMSE) on the training dataset. The model's predictions are saved for submission.

Approach and Methodology

Preprocessing

The first step in the workflow involves preprocessing the data. We perform the following tasks:

- Label Encoding: We use `LabelEncoder` from `sklearn` to encode the class labels.
- Feature Extraction: We use the `librosa` library to extract audio features such as MFCC, chroma, zero-crossing rate, and spectral contrast.
- SMOTE: For handling class imbalance, we use Synthetic Minority Oversampling Technique (SMOTE).

Model Training

We train a Random Forest Classifier using the features extracted from the audio files. Hyperparameter tuning is performed using GridSearchCV with cross-validation. The model is evaluated on both training and validation datasets.

Evaluation Metrics

The model is evaluated using:

- Classification Report: We evaluate precision, recall, and F1-score.
- RMSE: The Root Mean Squared Error (RMSE) is calculated on the training data to assess model performance.

Code Implementation

The following code is used to perform data preprocessing, feature extraction, model training, and evaluation.

Preprocessing the Data

```
import os
import pandas as pd
import numpy as np
import librosa
import joblib
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, mean_square
from imblearn.over_sampling import SMOTE

def preprocess_data(train_df, test_df):
    label_encoder = LabelEncoder()
    train_df['label'] = label_encoder.fit_transform(train_df['label'])
    test_df['label'] = -1  # Placeholder
    joblib.dump(label_encoder, 'label_encoder.pkl')
    return train_df, test_df, label_encoder
```

Feature Extraction

```
def extract_features(file_path, n_mfcc=13):
    try:
        y, sr = librosa.load(file_path, sr=None)
        mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
        mfccs_mean = np.mean(mfccs, axis=1)
        mfccs_std = np.std(mfccs, axis=1)
        chroma = librosa.feature.chroma_stft(y=y, sr=sr)
        chroma_mean = np.mean(chroma, axis=1)
        zcr = librosa.feature.zero_crossing_rate(y)
        zcr_mean = np.mean(zcr)
        spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr)
        contrast_mean = np.mean(spectral_contrast, axis=1)
        features = np.hstack([mfccs_mean, mfccs_std, chroma_mean, [zcr_mean], co
        return features
    except Exception as e:
        print(f"Error with {file_path}: {e}")
        return np.zeros(n_mfcc*2 + 12 + 1 + 7)
```

Model Training and Hyperparameter Tuning

```
# Load data
train_df = pd.read_csv('data/train.csv')
test_df = pd.read_csv('data/test.csv')

# Preprocess data
train_df, test_df, label_encoder = preprocess_data(train_df, test_df)
X_features = extract_all_features(train_df, is_train=True)
y_labels = train_df['label'].values

# Split and Normalize
X_train, X_val, y_train, y_val = train_test_split(X_features, y_labels, test_siz
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# SMOTE
smote = SMOTE(random_state=42, k_neighbors=1)
X_resampled, y_resampled = smote.fit_resample(X_train_scaled, y_train)

# Train with GridSearch
model = RandomForestClassifier(class_weight='balanced', random_state=42)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
grid_search = GridSearchCV(model, param_grid, cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_resampled, y_resampled)
best_model = grid_search.best_estimator_
```

Model Evaluation and RMSE Calculation

```
# Evaluate model
y_train_pred = best_model.predict(X_train_scaled)
rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
print(f"RMSE on Training Data: {rmse}")

# Print Classification Report
y_pred = best_model.predict(X_val_scaled)
print("Validation Results:")
print(classification_report(y_val, y_pred))
```

```
# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_val, y_pred))
```

Saving the Model

```
def save_model_and_scaler(model, scaler, model_path="model/random_forest_model.p
    os.makedirs("model", exist_ok=True)
    joblib.dump(model, model_path)
    joblib.dump(scaler, scaler_path)

save_model_and_scaler(best_model, scaler)
```

Test Prediction

```
def preprocess_test_only(test_df, label_encoder_path='label_encoder.pkl'):
    test_df['label'] = -1
    return test_df

# Load Model and Preprocess Test
model, scaler = load_model_and_scaler()
test_df = preprocess_test_only(test_df)
X_test = extract_all_features(test_df, is_train=False)
X_test_scaled = scaler.transform(X_test)
y_pred = model.predict(X_test_scaled)
decoded_predictions = label_encoder.inverse_transform(y_pred)

# Save to CSV
submission_df = test_df[['filename']].copy()
submission_df['label'] = decoded_predictions
submission_df.to_csv('predictions.csv', index=False)
print("    predictions.csv saved successfully.")
```

Conclusion

This workflow provided a comprehensive guide for training and evaluating a Random Forest model on an audio classification task. The key points included:

- Preprocessing audio data and extracting relevant features.
- Training a Random Forest classifier and tuning its hyperparameters.
- Evaluating the model using classification metrics and RMSE.
- Generating and saving the model and predictions.

The RMSE score on the training dataset was calculated to assess the model's performance.