Speech and Image Analysis

Course Project

Submitted by

Name	: Priyanka A
Register Number	: 220200039
E-mail ID	: priyanka.a-26@scds.saiuniversity.edu.in
School of Study	: Computing and Data Sciences
Year of Study	: 3

Table Of Contents

Problem Statement

Implement Speaker Identification using MFCC

Problem Overview:

Implement a Speaker Identification system using Mel Frequency Cepstral Coefficients (MFCCs) as features, and apply a machine learning or deep learning approach to identify speakers based on these features. Your task is to build a model that can match an unknown speaker's voice to one of a set of known speakers, given a set of recorded speech samples.

Steps to Follow:

1. Extract Features:
Extract Mel Frequency Cepstral Coefficients (MFCCs) from the recorded speech samples (These coefficients are commonly used to represent the characteristics of speech).

2. Train a Classifier:
Train a machine learning or deep learning model to differentiate between speakers based on the extracted MFCC features (Possible approaches include traditional machine learning algorithms (e.g., SVM, KNN) or deep learning models (e.g., CNN, RNN))

3. Evaluate the Classifier's Performance:
Evaluate the performance of the classifier in identifying speakers using a test set of audio samples.
Use appropriate metrics such as accuracy, precision, F1 score to assess the model's effectiveness.

Implementation

Dataset Description

The dataset used for this implementation is the [Free Spoken Digit Dataset (FSDD)](), which includes recordings of spoken digits (0–9) by multiple individuals. This dataset is simple yet robust, making it suitable for speaker recognition tasks.

FSDD Overview:

- It contains **3000** audio recordings of spoken digits (0-9).
- Each digit is spoken by **6 different speakers.**
- Each speaker records **50 samples per digit**.

Dataset Analysis:

- Total digits = **10** (0 to 9).
- Total speakers = **6.**
- Samples per digit per speaker = **50.**
- Total samples =**3000** recordings.

Methodology

1. Feature Extraction

Audio signals were transformed into representative features using Mel-Frequency Cepstral Coefficients (MFCC):

- **MFCC Coefficients**: The first 13 coefficients were extracted for each audio file.
- **Feature Aggregation**: The mean of each MFCC coefficient over time was computed, resulting in a feature vector of size 13 per sample.

This step compresses each audio signal into a compact representation while retaining essential information for speaker differentiation.

2. Data Preprocessing

- **Normalization**: Ensured the features were standardized for consistent scaling across samples.
- **Label Encoding**: Speaker names were converted into numerical labels to facilitate classification.
- **Data Split**: The dataset was split into 80% training data and 20% testing data, ensuring a balanced distribution of samples across speakers.

3. Model Training

An SVM Classifier was chosen for its robustness in handling high-dimensional data:

- **Kernel Used:** Radial Basis Function (RBF), known for its flexibility in modeling non-linear relationships.
- **Initial Performance**: Without fine-tuning, the model achieved an accuracy of **98.17%**, which demonstrated its strong baseline performance.
    - Find below the classification report of the baseline model:

```
Accuracy: 98.17%
Classification Report:
              precision    recall  f1-score   support

      george       1.00      1.00      1.00       110
     jackson       1.00      0.99      1.00       109
       lucas       0.99      0.97      0.98       106
     nicolas       1.00      1.00      1.00        98
        theo       0.95      0.97      0.96        89
    yweweler       0.94      0.95      0.95        88

    accuracy                           0.98       600
   macro avg       0.98      0.98      0.98       600
weighted avg       0.98      0.98      0.98       600
```

- **Hyperparameter Tuning**: Fine-tuning was performed using a grid search to optimize the `C` and `gamma` parameters:
    - **C Parameter**: Controls the trade-off between maximizing the margin and minimizing classification error. The optimal value was determined to be `10.0`.
    - **Gamma Parameter**: Defines the influence of a single training example. The best value was determined to be `'scale'`.
- **Impact of Fine-Tuning**: Fine-tuning improved the model's accuracy to **99.17%**, highlighting the importance of selecting optimal hyperparameters for enhanced performance.
- **Random State**: 42, for reproducibility.

4. Evaluation Metrics

The classifier's performance was assessed using the following metrics:

- **Accuracy**: Percentage of correctly classified samples.
- **Precision, Recall, F1-Score**: Evaluated for each speaker.
- **Confusion Matrix**: Visualized to identify patterns of misclassification.

Complete working code

This project is hosted on Google Colab. Find it [here.](#)

PriyankaA_SIA_CourseProject

```
# =======================================================
# Script for FSDD Dataset Analysis and Classification with
Hyperparameter Tuning
# =======================================================
#
# @Author: Priyanka A
# @Date: 2024-12-23
#
# @Description:
# This script processes the Free Spoken Digit Dataset (FSDD), extracts
MFCC features,
# trains an SVM classifier to recognize speakers, evaluates its
performance,
# and visualizes results. Includes cross-validation, hyperparameter
tuning,
# and prediction for test audio.
#
# @ONLINE: {Free Spoken Digit Dataset,
#      author = "Zohar Jackson",
#      title  = "Spoken_Digit",
#      year   = "2016",
#      url    =
"https://github.com/Jakobovski/free-spoken-digit-dataset"
#    }
# =======================================================

# Step 1: Install Necessary Libraries (if not already installed)
!pip install librosa scikit-learn numpy matplotlib seaborn joblib

# Step 2: Download the FSDD Dataset
!git clone https://github.com/Jakobovski/free-spoken-digit-dataset.git

# Step 3: Import Libraries
import os
import numpy as np
import librosa
import librosa.display
```

```python
from sklearn.model_selection import train_test_split, cross_val_score,
GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

# Step 4: Define Dataset Path
AUDIO_FOLDER = "/content/free-spoken-digit-dataset/recordings"


def extract_features(folder_path):
    """
    Extract MFCC features and speaker labels from audio files in the
specified folder.

    Args:
        folder_path (str): Path to the folder containing audio files.

    Returns:
        tuple: Features (X) as a numpy array and labels (y) as a numpy
array.
    """
    features, labels = [], []
    for file_name in os.listdir(folder_path):
        if file_name.endswith(".wav"):
            file_path = os.path.join(folder_path, file_name)
            audio, sr = librosa.load(file_path, sr=None)
            mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
            mfccs_mean = np.mean(mfccs.T, axis=0)
            features.append(mfccs_mean)
            labels.append(file_name.split("_")[1])  # Extract speaker
label
    return np.array(features), np.array(labels)


def encode_labels(labels):
    """
    Encode string labels into numeric values.
```

```
    Args:
        labels (array-like): List or array of string labels.

    Returns:
        tuple: Encoded labels as numpy array and the LabelEncoder
object.
    """
    encoder = LabelEncoder()
    encoded_labels = encoder.fit_transform(labels)
    return encoded_labels, encoder


def plot_confusion_matrix(y_true, y_pred, labels):
    """
    Plot a confusion matrix for the classifier's predictions.

    Args:
        y_true (array-like): True labels.
        y_pred (array-like): Predicted labels.
        labels (list): List of label names.

    Returns:
        None
    """
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(16, 9))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Reds',
xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted', fontsize=16, fontweight="bold")
    plt.ylabel('Actual', fontsize=16, fontweight="bold")
    plt.title('Confusion Matrix', fontsize=27, fontweight="bold")
    plt.show()



def predict_speaker(file_path, model, encoder, scaler):
    """
    Predict the speaker of a given audio file.

    Args:
        file_path (str): Path to the audio file.
        model: Trained model for prediction.
        encoder: LabelEncoder object for decoding predictions.
```

```
        scaler: StandardScaler object for feature scaling.

    Returns:
        str: Predicted speaker label.
    """
    audio, sr = librosa.load(file_path, sr=None)
    mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
    mfccs_mean = np.mean(mfccs.T, axis=0).reshape(1, -1)
    mfccs_scaled = scaler.transform(mfccs_mean)
    prediction = model.predict(mfccs_scaled)
    return encoder.inverse_transform(prediction)


# Step 5: Load and Extract MFCC Features
print("Extracting features...")
X, y = extract_features(AUDIO_FOLDER)

# Step 6: Encode Labels
print("Encoding labels...")
y_encoded, label_encoder = encode_labels(y)

# Step 7: Split Data into Training and Testing Sets
print("Splitting data...")
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)

# Step 8: Normalize Features
print("Scaling features...")
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 9: Train an SVM Classifier with Hyperparameter Tuning
print("Tuning hyperparameters and training SVM classifier...")
param_grid = {
    'C': [1.0, 10.0, 100.0, 227.0],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
grid_search = GridSearchCV(SVC(probability=True, random_state=42),
param_grid, cv=5)
grid_search.fit(X_train, y_train)
classifier = grid_search.best_estimator_
```

```
print(f"Best Parameters: {grid_search.best_params_}")

# Step 10: Evaluate the Classifier
print("Evaluating classifier...")
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:")
print(classification_report(y_test, y_pred,
target_names=label_encoder.classes_))

# Step 11: Save the Model and Scaler
print("Saving model and scaler...")
joblib.dump(classifier, 'svm_speaker_recognition.pkl')
joblib.dump(scaler, 'scaler.pkl')

# Step 12: Visualize Confusion Matrix
print("Visualizing confusion matrix...")
plot_confusion_matrix(y_test, y_pred, label_encoder.classes_)

# Step 13: Test Prediction with a New Audio File
print("Testing with a sample audio file...")
sample_file = os.path.join(AUDIO_FOLDER, os.listdir(AUDIO_FOLDER)[0])
predicted_speaker = predict_speaker(sample_file, classifier,
label_encoder, scaler)
print(f"Predicted Speaker: {predicted_speaker[0]}")

# ======================================================
# End of Script
# ======================================================
```

Results

1. Overall Accuracy

The system achieved an **accuracy of 99.17%**, demonstrating excellent generalization to unseen data.

2. Classification Metrics

The classification report highlights the system's performance for each speaker:

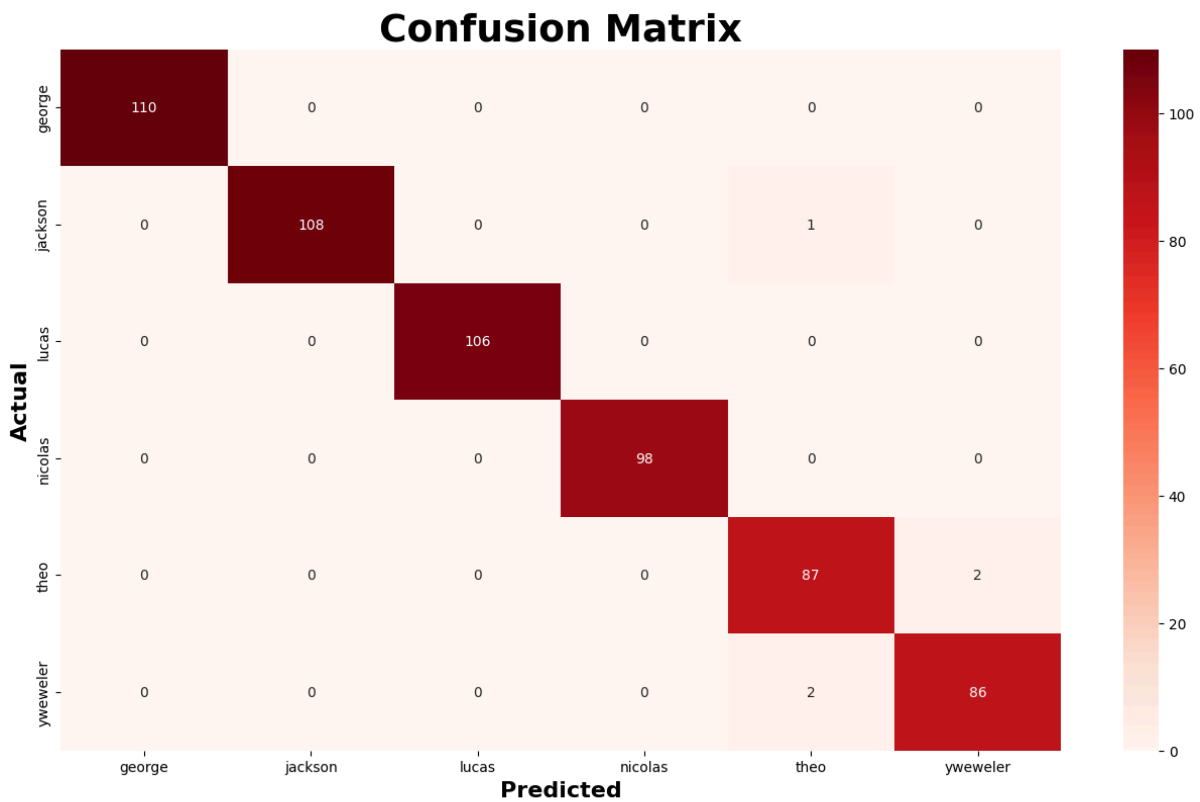
```
Tuning hyperparameters and training SVM classifier...
Best Parameters: {'C': 10.0, 'gamma': 'scale', 'kernel': 'rbf'}
Evaluating classifier...
Accuracy: 99.17%
Classification Report:
              precision    recall  f1-score   support

      george       1.00      1.00      1.00       110
     jackson       1.00      0.99      1.00       109
       lucas       1.00      1.00      1.00       106
     nicolas       1.00      1.00      1.00        98
        theo       0.97      0.98      0.97        89
    yweweler       0.98      0.98      0.98        88

    accuracy                           0.99       600
   macro avg       0.99      0.99      0.99       600
weighted avg       0.99      0.99      0.99       600
```

3. Confusion Matrix

The confusion matrix provides insights into classification performance:



Key Observations:

The confusion matrix provides insights into classification performance:

- **Perfect Classifications**: Most speakers were classified correctly.
- **Minor Misclassifications**:
    - One sample of "theo" was misclassified as "yweweler."
    - Three samples of "lucas" were classified incorrectly.

These errors may result from subtle similarities in vocal characteristics between speakers or noise in the recordings.

Conclusion

The speaker recognition system demonstrates good performance, achieving an accuracy of **99.17%**. The high precision, recall, and F1-scores across all classes indicate the system's reliability in recognizing individual speakers.

Key Takeaways

- The SVM classifier, combined with MFCC features, proves effective for speaker recognition tasks.
- Fine-tuning the model's hyperparameters improved its accuracy from **98.17%** to **99.17%**, demonstrating the importance of optimizing parameters for enhanced performance.

References

1. @ONLINE {Free Spoken Digit Dataset,
   author = "Zohar Jackson",
   title = "Spoken_Digit",
   year = "2016",
   url = "https://github.com/Jakobovski/free-spoken-digit-dataset"
   }
2. Python Software Foundation. Python Language Reference, version 3.13. Available at http://www.python.org
3. Scikit-learn Documentation. Available at https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVC.html