小组成员信息

组号：实验课 第 2 小组

组长：巩羽飞

组员：廉涟 李玉昆 王铭慧 王菁

贡献度：每人 20%

实验目标：

通过本实验，你将深入了解和实践说话人识别技术，并掌握利用声音特征进行有效说话人识别的基本方法，了解不同特征和模型对识别准确率的影响。
实验的核心目标是使用 TIMIT 数据集来训练一个说话人识别系统，涵盖数据预处理、特征提取、模型训练和评估等关键步骤。

1.实验准备

```
## 导入必要的库
import json
import matplotlib.pyplot as plt
import joblib
import scipy
import scipy.fftpack
from sklearn.mixture import GaussianMixture
import numpy
import scipy.io.wavfile
from scipy.fftpack import dct
import librosa
import numpy as np
from scipy.signal import lfilter
from scipy.fftpack import fft, ifft
```

2.数据预处理

```
def load_data_from_json(data_json):
    """从 JSON 文件中加载数据"""
    with open(data_json, 'r') as f:
        data = json.load(f)
    return data
```

```
def load_audio_data(filepath):
    """加载音频文件并返回 NumPy 数组类型的音频数据"""
    audio, _ = librosa.load(filepath, sr=None)
    return np.array(audio)
```

3. 特征提取

MFCC 模型，它是一种基于内耳频率分析的人类声音感知模型，用于音频信号处理的特征提取技术。它的主要作用是将原始音频信号转换为一组具有更好可分离性和可识别性的特征向量，方便后续的语音识别、说话人识别、语音合成和音频分类等应用。

• 原理：MFCC 模型基于滤波器组的频率分析，滤波器组的带宽间隔约为临界子带的间隔。由于频率较低的声音在内耳蜗基底膜上行波传递的距离大于频率较高的声音，因此低音容易掩蔽高音，而高音掩蔽低音较困难。因此，在低频处的声音掩蔽的临界带宽较高频要小。基于这一原理，人们从低频到高频这一段频带内按临界带宽的大小由密到疏安排一组带通滤波器，对输入信号进行滤波。将每个带通滤波器输出的信号能量作为信号的基本特征，对此特征经过进一步处理后就可以作为语音的输入特征。

• 模型结构：MFCC 特征提取的过程主要分为预加重、加窗、Mel 滤波器组、离散余弦变换（DCT）等步骤。预加重阶段对原始音频信号进行高通滤波，去除低频部分的噪声。加窗阶段对每一帧进行加窗处理，以避免由于信号突然截断而产生的频谱泄漏。Mel 滤波器组阶段将频率轴转换为 Mel 频率轴，并使用一组 Mel 滤波器对频谱进行滤波，得到 Mel 频率轴上的能量值。最后，通过 DCT 将滤波后的能量值转换为 MFCC 系数。

• 使用情况：MFCC 模型在语音识别、语音合成、说话人识别和音频分类等领域有广泛的应用。相对于其他特征提取方法，MFCC 具有良好的鲁棒性和较高的识别准确率。此外，由于 MFCC 特征不依赖于信号的性质，对输入信号不做任何的假设和限制，又利用了听觉模型的研究成果，因此这种参数比基于声道模型的 LPCC 相比具有更好的鲁棒性，更符合人耳的听觉特性，且当信噪比降低时仍然具有较好的识别性能。

```
def extract_features(audio):
    """提取音频特征（MFCC 特征）"""
    # y: 音频时间序列, sr: y 的采样率, n_mfcc: 要返回的 MFCC 数量
    mfccs = librosa.feature.mfcc(y=audio, sr=22050, n_mfcc=20)
    return mfccs

def main():
    """主函数"""
    train_data = load_data_from_json('train_info.json')
    test_data = load_data_from_json('test_info.json')

    train_features_dict = {}
    test_features_dict = {}
#  处理训练数据
    for item in train_data:
        speaker_id = item['speaker_id']
        if speaker_id not in train_features_dict:
            train_features_dict[speaker_id] = []
```

```
        # 调库实现 mfcc 特征提取：
        audio = load_audio_data(item['filepath'])
        features = extract_features(audio)

        # 将提取的特征添加到对应说话人的特征列表中。
        train_features_dict[speaker_id].append(features)

    for speaker_id in train_features_dict:
        # 将每个说话人的特征列表堆叠成一个二维数组，其中每一行是一个样本的特征向量
        train_features_dict[speaker_id] = np.vstack(train_features_dict[speaker_id])

    # 处理测试数据
    for item in test_data:
        speaker_id = item['speaker_id']
        if speaker_id not in test_features_dict:
test_features_dict[speaker_id] = []

        # 调库实现 mfcc 特征提取：
        audio = load_audio_data(item['filepath'])
        features = extract_features(audio)

        test_features_dict[speaker_id].append(features)

    for speaker_id in test_features_dict:
        test_features_dict[speaker_id] = np.vstack(test_features_dict[speaker_id])
```

4. 模型的选择与训练

我们首先采用 **GMM** 模型，GMM 的基本原理是将一个复杂的多维概率密度函数建模成多个高斯分布的线性组合。每个高斯分布都有自己的均值和协方差矩阵，它们共同描述了数据在该分布下的分布情况。具体来说，对于由 N 个样本组成的数据集，GMM 的基本假设是每个样本点都是从 K 个高斯分布混合而成的分布中独立地采样得到的。

模型结构

GMM 模型的结构主要包括以下几个部分：

• 高斯分布组件：模型中包含 K 个高斯分布组件，每个组件都有自己的均值向量、协方差矩阵和权重。这些参数共同定义了每个高斯分布的形状和位置。

• 权重：每个高斯分布组件都有一个权重，表示该组件在混合模型中的贡献程度。权重之和为 1，确保整个模型的概率密度函数在整个定义域上积分为 1。

在 GMM 模型中，数据点的生成过程可以描述为：首先根据权重选择一个高斯分布组件，然后从该组件对应的高斯分布中采样生成数据点。

使用情况

GMM 模型在多个领域都有广泛的应用

• 数值逼近：利用 GMM 模型可以逼近复杂的概率密度函数，实现对数据的精确描述。

•语音识别：GMM 模型可以用于语音信号的建模和识别，提高语音识别的准确性和鲁棒性。

• 图像分类与去噪：GMM 模型可以用于图像数据的建模和分类，同时也可用于图像去噪和重构，提高图像的质量和清晰度。

• 故障诊断：在故障诊断领域，GMM 模型可以用于异常检测和故障预测，帮助及时发现和解决潜在问题。
• 视频分析：GMM 模型可用于视频目标的跟踪和识别，提高视频分析的准确性和实时性。此外，GMM 模型还可用于密度估计、目标识别与跟踪等领域。通过不断调整和优化模型的参数，GMM 模型可以适应不同的数据集和任务需求，实现更精确和可靠的数据分析和处理。需要注意的是，GMM 模型的性能和效果受到多种因素的影响，包括数据的分布特性、模型的复杂度、参数的初始化方式以及优化算法的选择等。因此，在实际应用中需要根据具体情况进行调整和优化。

#GMM

```
def train_gmm_per_speaker(features_dict, num_components):
    gmms = {}
    for speaker_id, features in features_dict.items():
        # 初始化 GMM 模型，使用 diagonal covariance_type（对角协方差矩阵）
          gmm = GaussianMixture(n_components=num_components, covariance_type='diag', random_state=42)
        # 对特定说话人的特征数据进行训练
        gmm.fit(features)
        # 将训练好的 GMM 模型存储在字典中，键为说话人标识符，值为对应的 GMM 模型
        gmms[speaker_id] = gmm
    return gmms
```

然后我们又尝试 **LPCC 模型**，是一种基于线性预测编码（LPC）技术的语音特征提取方法。LPCC 主要用于语音信号处理和语音识别等领域，它结合了 LPC 分析和倒谱分析的优势，以提取出更具代表性的语音特征。
LPC 分析的基本思想是一个语音的取样可用过去若干语音取样的线性组合来逼近。其频率特性曲线会在共振峰频率处出现峰值，因此 LPC 可以看做一种短时谱估计法。LPCC 则是在 LPC 的基础上，进一步通过倒谱分析来提取特征。倒谱分析是一种在频域中将信号的谱转换为倒谱（cepstrum）的过程，能够分离出信号中的不同成分，并突出其中的周期性结构。
通过 LPCC 模型提取的特征，具有良好的可分离性和鲁棒性，对语音信号中的噪声和失真具有较强的抵抗能力。

#lpcc

```
def lpcc(path, order):
    sample_rate, signal = scipy.io.wavfile.read(path)
    # Pre-emphasis
    pre_emphasis = 0.97
    emphasized_signal = numpy.append(signal[0], signal[1:] - pre_emphasis * signal[:-1])
# 分帧
    frame_size = 0.025
    frame_stride = 0.01
    frame_length, frame_step = frame_size * sample_rate, frame_stride * sample_rate
    signal_length = len(emphasized_signal)
```

```
    frame_length = int(round(frame_length))
    frame_step = int(round(frame_step))
    num_frames = int(numpy.ceil(float(numpy.abs(signal_length - frame_length)) / frame_step))
    pad_signal_length = num_frames * frame_step + frame_length
    z = numpy.zeros((pad_signal_length - signal_length))
    pad_signal = numpy.append(emphasized_signal, z)
    indices = numpy.tile(numpy.arange(0, frame_length), (num_frames, 1)) + numpy.tile(
        numpy.arange(0, num_frames * frame_step, frame_step), (frame_length, 1)).T
    frames = pad_signal[indices.astype(numpy.int32, copy=False)]

    # 加窗
    frames *= numpy.hamming(frame_length)

    lpcc = np.zeros((num_frames, order))
    for i in range(num_frames):
        autocorr = lfilter([1], np.array([1, -pre_emphasis]), frames[i])
r = autocorr[:order + 1]

        a = np.zeros(order + 1)
        k = np.zeros(order)
        a[0] = 1
        E = r[0]
        for m in range(1, order + 1):
            k[m - 1] = -np.sum(a[1:m] * r[m - 1:0:-1]) / E
            a[m] = k[m - 1]
            for j in range(1, m):
                a[j] += k[m - 1] * a[m - j]
            E *= 1 - k[m - 1] ** 2

        # 异常处理
        logE = np.log(E)
        if np.isnan(logE) or np.isinf(logE):
            # 当 logE 为 NaN 或无穷大时，将 lpcc 设置为全零向量
            lpcc[i] = np.zeros(order)
        else:
            lpcc[i] = fft(logE + ifft(np.log(np.abs(fft(a, 512)))[:order].real)).real

return lpcc
```

最后我们尝试 **softmax** 模型，它的核心思想是将每个输入变量投影到一个维度上，然后通过一个指数函数，把这个维度映射到(0,1)区间内，作为模型预测的结果。具体来说，softmax 函数接收一个输入数组（通常是模型的原始输出），然后将数组中的每个元素转换为正数，并且这些正数的总和为 1。这使得转换后的数组可以解释为一组概率分布，其中每个元素表示输入样本属于相应类别的概率。

模型结构

softmax 模型通常作为神经网络（特别是深度学习网络）的最后一层，用于将网络的原始输出转换为概率分布。神经网络的前几层通常负责提取输入数据的特征，而 softmax 层则负责将这些特征转换为类别概率。
在使用 softmax 模型进行分类时，我们通常会选择概率最大的类别作为预测结果。此外，由于 softmax 模型输出了每个类别的概率，我们还可以根据这些概率来评估模型的不确定性或置信度。
使用情况
softmax 模型在多种场景下都有广泛的应用，特别是在需要处理多类别分类问题的领域。以下是一些具体的使用情况：
• 图像识别：在图像识别任务中，softmax 模型常用于将卷积神经网络（CNN）的输出转换为不同类别的概率分布。例如，在识别手写数字或识别不同种类的图像时，softmax 模型可以帮助我们确定图像最可能属于的类别。
• 自然语言处理：在自然语言处理中，softmax 模型常用于文本分类、词性标注等任务。例如，在情感分析中，softmax 模型可以帮助我们确定文本表达的情感是正面的、负面的还是中性的。
• 语音识别：在语音识别系统中，softmax 模型可以用于将音频信号转换为不同单词或短语的概率分布，从而实现语音到文本的转换。

```
# 训练 Logistic 回归模型
model = LogisticRegression(max_iter=1000, multi_class='multinomial', solver='lbfgs')
model.fit(train_features_scaled, train_labels)

# 预测
test_predictions = model.predict(test_features_scaled)
```

5. 评价指标(准确率）

```
def calculate_accuracy(test_data, gmms, test_features_dict):
    """计算准确率，使用总得分来确定最可能的说话人"""
    correct_predictions = 0
    total_predictions = 0

    for item in test_data:
        speaker_id = item['speaker_id']
        features = test_features_dict[speaker_id]

        # 计算每个模型的得分总和
            individual_scores = {sid: gmm.score_samples(features).sum() for sid, gmm in gmms.items()}

        # 找出得分最高的说话人 ID 作为预测结果
```

```
        predicted_speaker = max(individual_scores, key=individual_scores.get)

        # 检查预测是否正确
if predicted_speaker == speaker_id:
            correct_predictions += 1

        total_predictions += 1

    # 计算准确率
    accuracy = correct_predictions / total_predictions if total_predictions > 0 else 0
    return accuracy
```

6. 分析和可视化

```
def visualize(audio, mfccs):
    """可视化音频波形和 MFCC 特征"""
    # 绘制音频波形
    plt.figure(figsize=(10, 4))
    plt.plot(audio)
    plt.title('Waveform')
    plt.xlabel('Sample')
    plt.ylabel('Amplitude')
    plt.show()

    # 绘制 MFCC 特征
    plt.figure(figsize=(10, 4))
    librosa.display.specshow(mfccs, x_axis='time')
    plt.colorbar()
    plt.title('MFCC')
    plt.tight_layout()
    plt.show()
```

7. 结果

Gmm_v1

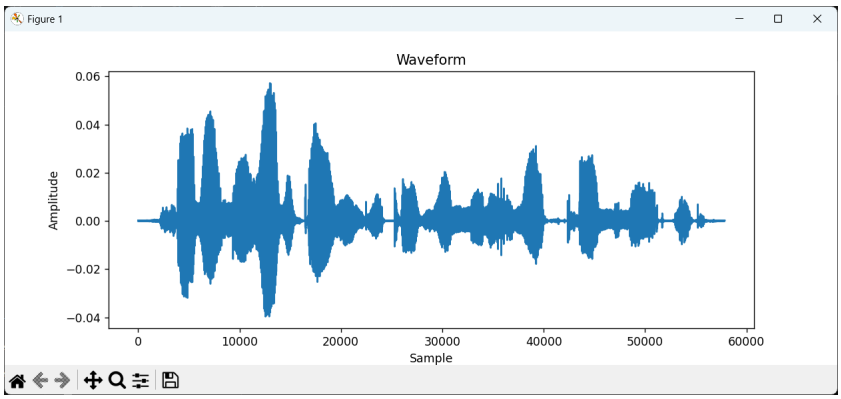


Figure 1
Waveform
Amplitude
Sample
0.06
0.04
0.02
0.00
−0.02
−0.04
0
10000
20000
30000
40000
50000
60000

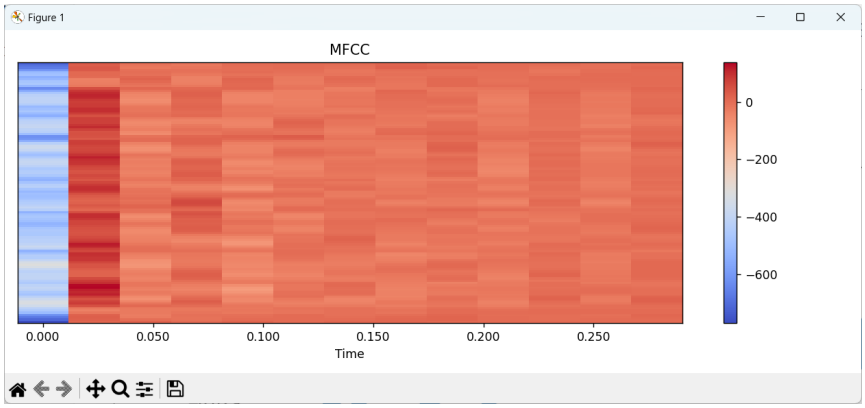


Figure 1
MFCC
Time
0.000
0.050
0.100
0.150
0.200
0.250
0
−200
−400
−600

Gmm_v2

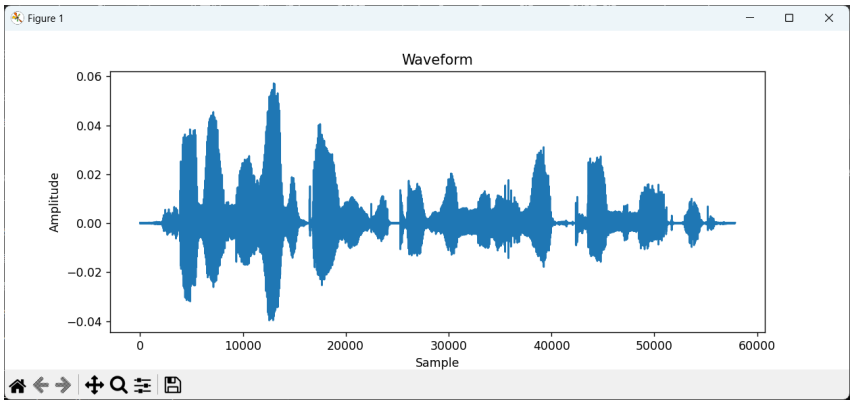


Figure 1
Waveform
Amplitude
Sample
0.06
0.04
0.02
0.00
−0.02
−0.04
0
10000
20000
30000
40000
50000
60000

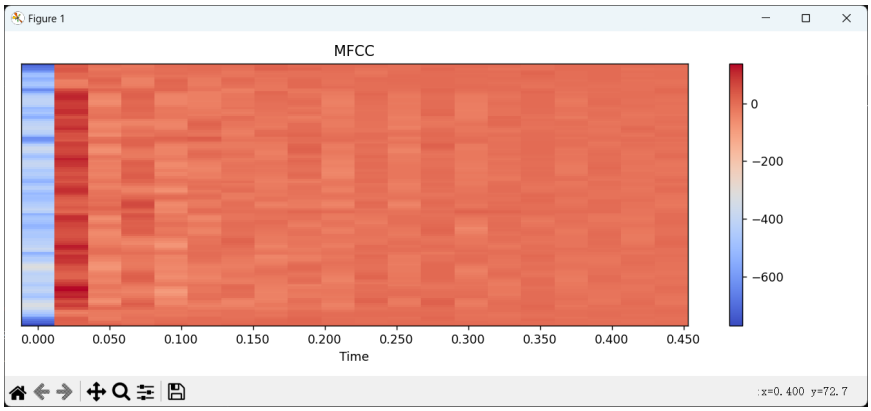
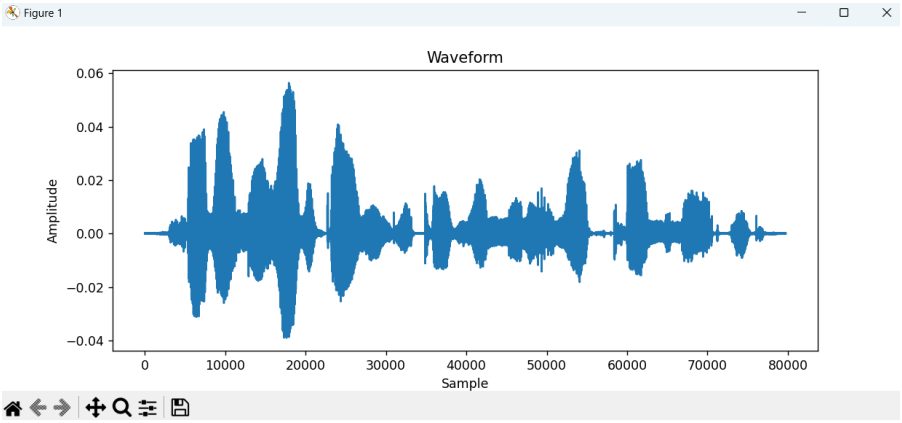
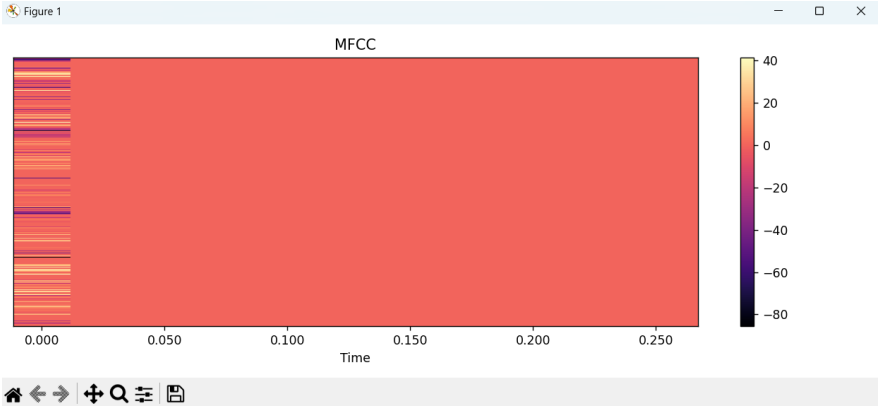


Figure 1
MFCC
Time
0.000
0.050
0.100
0.150
0.200
0.250
0.300
0.350
0.400
0.450
0
−200
−400
−600
x=0.400 y=72.7

Lpcc+Mfcc_v2





Softmax_v2