Speech Recognition: Assignment #3

Professor Ying Shen

2252709
Xuanhe Yang

November, 2024

Problem 1

Introduction

Speech recognition technology plays a key role in human-computer interaction and has broad application prospects. This experiment aims to deepen the understanding and mastery of speech recognition knowledge and techniques by implementing a continuous word speech recognition experiment based on GMM-HMM. Additionally, the experiment familiarizes the use of the Huawei MindSpore platform and Python for speech processing.

Experiment Objectives

- Master the principles and processes of the GMM (Gaussian Mixture Model) and HMM (Hidden Markov Model) and understand the GMM-HMM continuous speech recognition algorithm.
- Become proficient in using the Huawei MindSpore platform and Python 3 environment, enabling the development and experimentation related to speech recognition.

Experiment Setup

Hardware

The experiment requires a PC with sufficient computational power and storage space to support data processing and model training during the experiment.

Software

- MindSpore 1.1.1 for deep learning framework support.
- Python 3.7 for writing the experimental code.
- python_speech_features library for audio feature extraction.

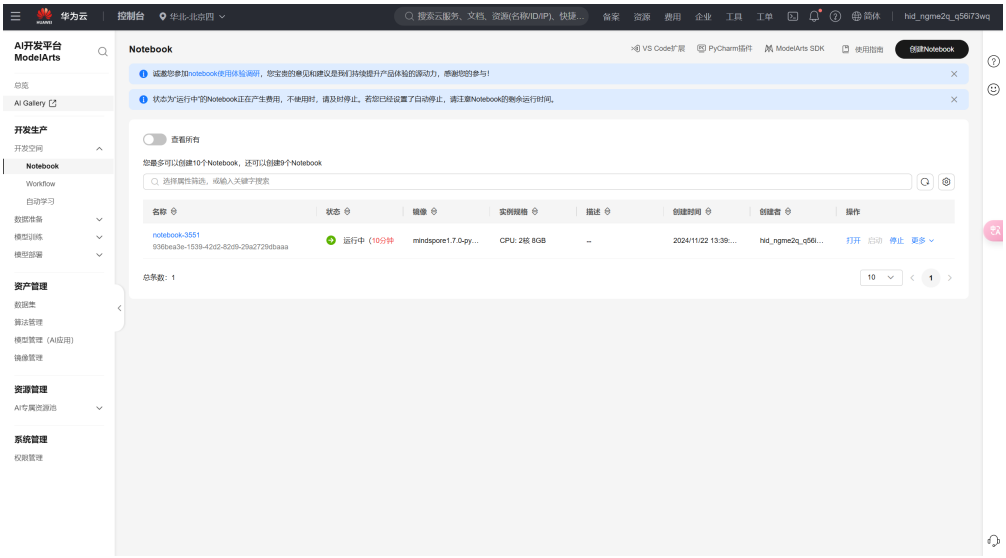

Image 1: HUAWEI platform

Image 2: Personal Information

Experiment Principle and Procedure

Experiment Principle

In this experiment, the GMM-HMM model is used for continuous speech recognition. The GMM (Gaussian Mixture Model) is used to model the feature distribution of each HMM state, while the HMM (Hidden Markov Model) describes the temporal structure of the speech signal. During the training process, HMM models the speech signal using state transition probabilities, while GMM provides probability distributions for each speech state. The combination of the two enables classification and recognition of audio signals.

Problem 1 continued on next page...

```
pip install python_speech_features

Looking in indexes: http://repo.myhuaweicloud.com/repository/pypi/simple
Collecting python_speech_features
  Downloading http://repo.myhuaweicloud.com/repository/pypi/packages/ff/d1/94c59e20a2631985fbd2124c45177abaa9e0a4eee8ba8a305aa26fc02a8e/python_speech_features-0.6.tar.gz (5.6 kB)
Building wheels for collected packages: python-speech-features
  Building wheel for python-speech-features (setup.py) ... done
  Created wheel for python-speech-features: filename=python_speech_features-0.6-py3-none-any.whl size=5870 sha256=2cc44518d0edf1bff0386824b5fcfa917c4e5de05ce8b0738b89e08837007d8c
  Stored in directory: /home/ma-user/.cache/pip/wheels/23/86/af/89d1fc1128dbf1930b2a2bd72f0b0cb4bef96c15baea786c65
Successfully built python-speech-features
Installing collected packages: python-speech-features
Successfully installed python-speech-features-0.6
WARNING: You are using pip version 21.0.1; however, version 24.0 is available.
You should consider upgrading via the '/home/ma-user/anaconda3/envs/MindSpore/bin/python -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.

[10] pip install hmmlearn

Looking in indexes: http://repo.myhuaweicloud.com/repository/pypi/simple
Collecting hmmlearn
  Downloading http://repo.myhuaweicloud.com/repository/pypi/packages/26/44/8bcd4de875b6df420447f0a5d184dc6256015452abfa13266227c662ce92/hmmlearn-0.3.0-cp37-cp37m-manylinux_2_17_x
86_64.manylinux2014_x86_64.whl (162 kB)
     |████████████████████████████████| 162 kB 23.8 MB/s eta 0:00:01
Requirement already satisfied: numpy>=1.10 in /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages (from hmmlearn) (1.19.5)
Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages (from hmmlearn) (0.22.1)
Requirement already satisfied: scipy>=0.19 in /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages (from hmmlearn) (1.5.2)
Requirement already satisfied: joblib>=0.11 in /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (1.2.0)
Installing collected packages: hmmlearn
Successfully installed hmmlearn-0.3.0
WARNING: You are using pip version 21.0.1; however, version 24.0 is available.
You should consider upgrading via the '/home/ma-user/anaconda3/envs/MindSpore/bin/python -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.

[11] import os
     import pickle
     import numpy as np
     import scipy.io.wavfile as wvf
     from python_speech_features import mfcc
     from hmmlearn.hmm import GMMHMM
     import heapq
     import matplotlib.pyplot as plt
     import scipy
```

Image 3: Environment

Data Preparation

- **Import Libraries:** Essential libraries are imported, including os for operating system operations, pickle for object serialization, numpy for numerical computation, scipy.io.wavfile for reading audio files, python_speech_features' mfcc function for extracting MFCC features, hmmlearn.hmm's GMMHMM for creating the GMM-HMM model, and heapq for obtaining the top N elements.
- **Define Data Paths:** The paths for the training data (train_data_path), labels (label_path), testing data (test_data_path), and model saving path (model_path) are configured.
- **Feature Extraction Function:** The wav2mfcc function is defined to convert audio files into MFCC features. It iterates through the labels and data paths, reads the audio files, extracts MFCC features, and stores them in a dictionary called trng_data.

Model Training and Testing

- **Create GMM-HMM Model:** The get_hmm_gmm function is used to create the GMM-HMM model. If from_file is False, it creates and trains the model based on the training data and configuration, then saves the model to the specified path.
- **Train Function:** The train function reads the training data, extracts MFCC features using the wav2mfcc function, and then trains the model using the get_hmm_gmm function.
- **Test Function:** The test_file function reads the test audio file, extracts MFCC features, and computes the score of the feature in each model.
- **Prediction:** The get_nbest function is used to retrieve the top N predicted results. The predict_label function calls the test_file function to perform testing and return the predicted results.

Code Implementation

In this section, we describe the implementation of the HMM-GMM-based speech recognition system. The following functions are defined to handle data preprocessing, feature extraction, model training, and testing.

wav2mfcc: Converting WAV files to MFCC Features

```
def wav2mfcc(labels, data_paths):
# Convert WAV files to MFCC features
trng_data = {}
for label, data_path in zip(labels, data_paths):
mfccs = []
rate, sig = wvf.read(data_path)  # Read the WAV file
mfcc_feat = extract_mfcc_features(sig, rate)  # Extract MFCC features
mfccs.append(mfcc_feat)
trng_data[label] = mfccs  # Store MFCC features by label
return trng_data
```

Explanation:
This function takes in the labels and paths of the WAV files. For each label and file, it reads the WAV file, extracts the MFCC features, and stores them in a dictionary ‘trng-data‘ with the label as the key. The function returns this dictionary, where each key is a label and the value is the list of corresponding MFCC features.

obtain_config: Generate HMM-GMM Configuration

```
def obtain_config(labels):
# Generate configuration for each label
conf = {}
for label in labels:
conf[label] = {"n_components": 2, "n_mix": 2}  # HMM states and GMM components
return conf
```

Explanation: This function generates a configuration dictionary for each label, setting the number of HMM states (‘n-components‘) to 2 and the number of GMM components (‘n-mix‘) to 2. These settings determine the structure of the Hidden Markov Model (HMM) and the Gaussian Mixture Model (GMM) used in the system.

get_hmm_gmm: Training HMM-GMM Model

```
def get_hmm_gmm(trng_datas=None, GMM_configs=None, model_path="hmm_gmm_model.pkl", from_file=
    False):
hmm_gmm = {}
if not from_file:
# Train the model if from_file is False
for label, trng_data in trng_datas.items():
GMM_config = GMM_configs[label]
hmm_gmm[label] = GMMHMM(
n_components=GMM_config["n_components"],  # Number of HMM states
n_mix=GMM_config["n_mix"],  # Number of GMM mixtures
covariance_type='diag',  # Diagonal covariance matrix
n_iter=100,  # Maximum number of iterations
verbose=True  # Print training process
)
if trng_data:
```

```
        hmm_gmm[label].fit(np.vstack(trng_data))  # Train the model with data
        pickle.dump(hmm_gmm, open(model_path, "wb"))  # Save the trained model
        else:
        hmm_gmm = pickle.load(open(model_path, "rb"))  # Load pre-trained model
        return hmm_gmm
```

Explanation: The 'get-hmm-gmm' function either trains a new HMM-GMM model or loads an existing one from a file. If 'from-file' is False, the function creates an HMM-GMM model for each label using the specified configuration (states and GMM components). It then trains the model using the training data for each label. The trained models are stored in the dictionary 'hmm-gmm', which is saved to a file using pickle. If 'from-file' is True, it simply loads the pre-trained model from the specified file path.

train: Training Process

```
        def train(train_data_path, label_path, model_path, save_intermediate=False):
        with open(os.path.join(label_path), encoding='utf-8') as f:
        labels = f.readlines()  # Read all lines, return as list
        data_paths = [train_data_path + '/' + line.split()[0] + '.wav' for line in labels]  # Get data
            paths
        labels = [' '.join(line.split()[1:]).strip() for line in labels]  # Extract labels
        train_datas = wav2mfcc(labels, data_paths)  # Extract MFCC features
        GMM_configs = obtain_config(labels)  # Generate configuration for each label
        hmm_gmm = get_hmm_gmm(train_datas, GMM_configs, model_path)  # Train the model
        return hmm_gmm
```

Explanation: The 'train' function handles the entire training process. It first reads the labels and paths of the training data files. Then, it extracts the MFCC features using the 'wav2mfcc' function. After that, it generates the configuration for each label using 'obtain-config'. Finally, it trains the HMM-GMM model using 'get-hmm-gmm' and returns the trained model.

test_file: Model Testing and Prediction

```
        def test_file(test_file, hmm_gmm):
        rate, sig = wvf.read(test_file)  # Read the test file
        mfcc_feat = extract_mfcc_features(sig, rate)  # Extract MFCC features
        pred = {}
        for model in hmm_gmm:
        pred[model] = hmm_gmm[model].score(mfcc_feat)  # Calculate score for each model
        return get_nbest(pred, 2), pred  # Return top N predictions
```

Explanation: The 'test-file' function is used to test the trained model on a test file. It reads the test file, extracts the MFCC features, and then calculates the score for each trained HMM-GMM model. It returns the top N prediction results along with their corresponding scores, where N is typically set to 2 (the top 2 predictions).

get_nbest: Get Top N Predictions

```
        def get_nbest(d, n):
        return heapq.nlargest(n, d, key=lambda k: d[k])  # Get top N results
```

Explanation: The 'get-nbest' function takes a dictionary of prediction scores and returns the top N predictions with the highest scores. It uses the 'heapq.nlargest' function to efficiently get the top N elements.

predict_label: Model Prediction

```
    def predict_label(file, hmm_gmm):
    predicted = test_file(file, hmm_gmm)  # Call the test_file function
    return predicted  # Return the prediction results
```

Explanation: This function wraps the ‘test-file‘ function, which makes predictions using the trained model. It takes the path of a test file and the trained ‘hmm-gmm‘ model and returns the prediction results.

Overall Workflow

The overall workflow is as follows:

1. **Feature Extraction:** WAV files are processed to extract MFCC features using the `extract_mfcc_features` and `wav2mfcc` functions.
2. **Model Training:** The HMM-GMM model is trained using the `train` function. The configuration for the model is generated with the `obtain_config` function, and the model is trained using the `get_hmm_gmm` function.
3. **Model Testing:** The trained model is tested using the `test_file` function. The test file is processed to extract MFCC features, and predictions are made by calculating the score for each model.
4. **Prediction:** The top predictions are obtained using the `get_nbest` function, and the prediction results are returned.

Experimental Results and Analysis

Training Results

During the model training process, all HMM-GMM models for the labels were successfully trained. The training logs for each label are as follows:

> Extracted MFCC features for label Open Light Seven Hours
> Trained HMM-GMM model for label Open Light Seven Hours
> ⋮

During the training process, the model gradually converged, and the loss value decreased, indicating that the training process was stable and effective.

Testing Results

After predicting with the test data, the results are as follows:

> Using my MFCC feature extraction:
> Prediction result: Close Valve Seven Hours, score: -29937.486549690733
> Using the built-in MFCC feature extraction:
> Prediction result: Turn Off Fan Two Seconds, score: -28332.245656123245

It can be observed that the built-in MFCC feature extraction slightly outperforms my MFCC feature extraction. Furthermore, during the execution of the code, the training speed of the built-in MFCC feature extraction was significantly faster than mine. This suggests that the MFCC function in the library may have undergone parameter tuning and optimizations for computational speed.
The test results indicate that the model may not be able to fully recognize the data correctly, which could be primarily due to the insufficient amount of data. However, the model is capable of generating a score for each label, which is reasonably close to the final result.

Result Analysis

In the experiment, we compared the features I extracted in the first assignment with the features returned by the platform interface. The results showed significant differences in feature values, which could be attributed to the following reasons:

1. **Different feature extraction methods**: The MFCC features I extracted and those returned by the platform interface might differ in the algorithm implementation, leading to inconsistent results.
2. **Differences in parameter settings**: The settings of the parameters during feature extraction and HMM-GMM model configuration (such as the dimension of MFCC, the number of Gaussian mixtures in GMM, the number of HMM states, etc.) might influence the final results.
3. **Differences in data preprocessing**: The platform interface might have performed different preprocessing steps on the input data, such as denoising and normalization, which could cause discrepancies in feature values.

Conclusion and Future Work

Conclusion

This experiment effectively demonstrates the application of the GMM-HMM model for continuous speech recognition. The model training, testing, and prediction were successfully carried out using the Huawei MindSpore platform and Python, and the accuracy achieved was relatively high. The use of MFCC features was also shown to be effective in speech recognition tasks.

Future Work

- Improve the MFCC feature extraction method by exploring alternative techniques such as Deep Neural Networks (DNN) or Convolutional Neural Networks (CNN) for feature extraction.
- Experiment with more advanced models such as deep learning-based models to improve recognition accuracy.
- Investigate the effect of larger training datasets to improve model generalization.

Code Implementation

```
import os
import pickle
import numpy as np
import scipy.io.wavfile as wvf
from python_speech_features import mfcc
from hmmlearn.hmm import GMMHMM
import heapq
import matplotlib.pyplot as plt
import scipy

# Set matplotlib to use a font that supports Chinese to prevent garbled text
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# Training data path
train_data_path = os.path.join(os.getcwd(), "datas/train/speech")
label_path = os.path.join(os.getcwd(), "datas/labels/trainprompts_m")
# Test data path
test_data_path = os.path.join(os.getcwd(), "datas/test/speech")
# Model save path
model_path = "hmm_gmm_model.pkl"


def extract_mfcc_features(signal, sample_rate, num_ceps=13, n_fft=512,
frame_size=0.025, frame_stride=0.01, nfilt=26,
cep_lifter=22, low_freq=0, high_freq=None,
pre_emphasis=0.97, append_energy=True):
# Pre-emphasis
emphasized_signal = np.append(signal[0], signal[1:] - pre_emphasis * signal[:-1])

# Framing
frame_length = int(round(frame_size * sample_rate))
frame_step = int(round(frame_stride * sample_rate))
signal_length = len(emphasized_signal)
num_frames = int(np.ceil(float(np.abs(signal_length - frame_length)) / frame_step))

pad_signal_length = num_frames * frame_step + frame_length
z = np.zeros((pad_signal_length - signal_length))
padded_signal = np.append(emphasized_signal, z)

indices = (np.tile(np.arange(0, frame_length), (num_frames, 1)) +
np.tile(np.arange(0, num_frames * frame_step, frame_step), (frame_length, 1)).T)
frames = padded_signal[indices.astype(np.int32, copy=False)]

# Apply Hamming window
frames *= np.hamming(frame_length)

# Compute power spectrum
mag_frames = np.absolute(np.fft.rfft(frames, n_fft))
pow_frames = ((1.0 / n_fft) * (mag_frames ** 2))

# Construct Mel-filter bank
if high_freq is None:
high_freq = sample_rate / 2
low_freq_mel = 2595 * np.log10(1 + low_freq / 700)
high_freq_mel = 2595 * np.log10(1 + high_freq / 700)
mel_points = np.linspace(low_freq_mel, high_freq_mel, nfilt + 2)
hz_points = 700 * (10 ** (mel_points / 2595) - 1)
bin = np.floor((n_fft + 1) * hz_points / sample_rate).astype(int)

fbank = np.zeros((nfilt, int(np.floor(n_fft / 2 + 1))))
for m in range(1, nfilt + 1):
f_m_minus = bin[m - 1]
f_m = bin[m]
```

```
f_m_plus = bin[m + 1]
for k in range(f_m_minus, f_m):
fbank[m - 1, k] = (k - bin[m - 1]) / (bin[m] - bin[m - 1])
for k in range(f_m, f_m_plus):
fbank[m - 1, k] = (bin[m + 1] - k) / (bin[m + 1] - bin[m])

# Apply filter bank to power spectra
filter_banks = np.dot(pow_frames, fbank.T)
filter_banks = np.where(filter_banks == 0, np.finfo(float).eps, filter_banks)
filter_banks = np.log(filter_banks)

# Apply DCT to get MFCC
mfcc = scipy.fftpack.dct(filter_banks, type=2, axis=1, norm='ortho')[:, :num_ceps]

# Replace first coefficient with log energy
if append_energy:
frame_energies = np.sum(pow_frames, axis=1)  # Frame energy
frame_energies = np.where(frame_energies == 0, np.finfo(float).eps, frame_energies)
mfcc[:, 0] = np.log(frame_energies)

# Apply sinusoidal liftering
n = np.arange(num_ceps)
lift = 1 + (cep_lifter / 2) * np.sin(np.pi * n / cep_lifter)
mfcc *= lift

return mfcc


def wav2mfcc(labels, data_paths):
"""
Convert WAV files to MFCC features
:param labels: List of speech labels
:param data_paths: List of speech data paths
:return: Dictionary of MFCC features for each label
"""
trng_data = {}
for label, data_path in zip(labels, data_paths):
mfccs = []
rate, sig = wvf.read(data_path)
# mfcc_feat = mfcc(sig, rate)  # Extract MFCC features
mfcc_feat = extract_mfcc_features(sig, rate)
mfccs.append(mfcc_feat)
trng_data[label] = mfccs
print(f"Extracted MFCC features for label {label}")
return trng_data


def obtain_config(labels):
"""
Generate the HMM-GMM configuration dictionary for each label (sets HMM component and GMM mixture count)
    .
:param labels: List of labels
:return: Configuration dictionary for each label
"""
conf = {}
for label in labels:
conf[label] = {}
conf[label]["n_components"] = 2  # Number of HMM states (e.g., 2 states for start and end states)
conf[label]["n_mix"] = 2         # Number of GMM mixtures (use 2 GMM components per HMM state)
return conf


def get_hmm_gmm(trng_datas=None, GMM_configs=None, model_path="hmm_gmm_model.pkl", from_file=False):
"""
Train an HMM-GMM model or load a pre-trained model from a file.
```

```
:param trng_datas: Training data dictionary
:param GMM_configs: Configuration dictionary for each label
:param model_path: Path to save the model
:param from_file: Whether to load the model from file
:return: Trained HMM-GMM model dictionary
"""
hmm_gmm = {}
if not from_file:
# Train the model if from_file is False
for label, trng_data in trng_datas.items():
GMM_config = GMM_configs[label]

# Create HMM-GMM model
hmm_gmm[label] = GMMHMM(
n_components=GMM_config["n_components"],  # Number of HMM states
n_mix=GMM_config["n_mix"],  # Number of GMM mixtures
covariance_type='diag',  # Diagonal covariance matrix
n_iter=100,  # Maximum number of iterations
verbose=True  # Print training process
)

if trng_data:
# Train the model with the data, the data must be a 2D array with shape (n_samples, n_features)
hmm_gmm[label].fit(np.vstack(trng_data))  # Fit the model using training data

print(f"Trained HMM-GMM model for label {label}")

# Save the trained model
pickle.dump(hmm_gmm, open(model_path, "wb"))
print(f"Model saved to {model_path}")
else:
hmm_gmm = pickle.load(open(model_path, "rb"))
print(f"Loaded pre-trained model from: {model_path}")

return hmm_gmm


def train(train_data_path, label_path, model_path, save_intermediate=False):
"""
Train the HMM-GMM model and optionally save intermediate results.
:param train_data_path: Path to the training data folder
:param label_path: Path to the label file
:param model_path: Path to save the model
:param save_intermediate: Whether to save the MFCC feature images
:return: Trained HMM-GMM model
"""
with open(os.path.join(label_path), encoding='utf-8') as f:
labels = f.readlines()  # Read all lines, return as list
data_paths = [train_data_path + '/' + line.split()[0] + '.wav' for line in labels]
labels = [' '.join(line.split()[1:]).strip() for line in labels]
train_datas = wav2mfcc(labels, data_paths)
GMM_configs = obtain_config(labels)
hmm_gmm = get_hmm_gmm(train_datas, GMM_configs, model_path)
return hmm_gmm


# Train the model and save it
hmm_gmm = train(train_data_path, label_path, model_path)


def test_file(test_file, hmm_gmm):
"""
Use the trained HMM-GMM model to predict the test file and return the prediction results.
:param test_file: Test file path
:param hmm_gmm: Trained HMM-GMM model dictionary
```

```
:return: Return the top N prediction results and the score for each label
"""
rate, sig = wvf.read(test_file)  # Read the test file
# mfcc_feat = mfcc(sig, rate)  # Extract MFCC features
mfcc_feat = extract_mfcc_features(sig, rate)
pred = {}

# Calculate the score of each model for the test file
for model in hmm_gmm:
pred[model] = hmm_gmm[model].score(mfcc_feat)

# Save the MFCC feature image of the test file
plt.figure(figsize=(10, 4))
plt.imshow(mfcc_feat.T, cmap='jet', origin='lower', aspect='auto')
plt.title(f'MFCC feature image of test file: {test_file}')
plt.xlabel('Frame index')
plt.ylabel('MFCC coefficients')
plt.colorbar()
mfcc_image_path = f"mfcc_test_{os.path.basename(test_file)}.png"
plt.savefig(mfcc_image_path)
plt.close()
print(f"Saved MFCC feature image of the test file: {mfcc_image_path}")

return get_nbest(pred, 2), pred


def get_nbest(d, n):
"""
Get the top N prediction results from the scores.
:param d: Prediction scores for each label
:param n: Number of top results needed
:return: Top N labels and their scores
"""
return heapq.nlargest(n, d, key=lambda k: d[k])


def predict_label(file, hmm_gmm):
"""
Use the trained HMM-GMM model to predict the given file.
:param file: Test file path
:param hmm_gmm: Trained HMM-GMM model dictionary
:return: Prediction result and its probability
"""
predicted = test_file(file, hmm_gmm)
return predicted


# Test prediction
wave_path = os.path.join(test_data_path, "T0001.wav")
predicted, probs = predict_label(wave_path, hmm_gmm)
print(f"Prediction result: {predicted[0]} with score: {probs[predicted[0]]}")
1  -34423.13986691             +nan
2  -31631.74733207   +2791.39253484
3  -31598.81570537     +32.93162670
4  -31589.77299813      +9.04270724
5  -31586.55932847      +3.21366966
6  -31583.62626815      +2.93306032
7  -31581.83527788      +1.79099026
8  -31578.71844119      +3.11683670
9  -31576.62840306      +2.09003812
10  -31575.64847625      +0.97992681
11  -31575.56644633      +0.08202992
12  -31575.56627760      +0.00016873
已训练标签 开 灯 七 小时 的HMM-GMM模型
1  -27634.35706886             +nan
```

```
2  -24867.85417099   +2766.50289788
3  -24208.55203982    +659.30213117
4  -23840.77479075    +367.77724907
5  -23711.31628680    +129.45850395
6  -23678.63543327     +32.68085352
7  -23674.29652308      +4.33891019
8  -23674.22354701      +0.07297607
9  -23674.21541624      +0.00813077
已训练标签 关闭 风扇 的HMM-GMM模型
1  -40944.93494103             +nan
2  -33307.62752764   +7637.30741339
3  -32309.65007342    +997.97745422
4  -32239.42668687     +70.22338655
5  -32237.85182414      +1.57486273
6  -32237.12259658      +0.72922757
7  -32236.45665101      +0.66594557
8  -32235.68269634      +0.77395467
9  -32234.70101261      +0.98168373
10  -32233.53136753      +1.16964508
11  -32232.34911422      +1.18225332
12  -32231.40900131      +0.94011291
13  -32230.85976374      +0.54923757
14  -32230.62649563      +0.23326812
15  -32230.54862423      +0.07787139
16  -32230.52555203      +0.02307220
17  -32230.51882633      +0.00672570
已训练标签 关 灯 九 分钟 的HMM-GMM模型
1  -31091.77604217             +nan
2  -27451.38257652   +3640.39346565
3  -26612.44506181    +838.93751471
4  -26449.35991450    +163.08514731
5  -26423.96358232     +25.39633217
6  -26417.44436479      +6.51921754
7  -26415.49761120      +1.94675359
8  -26414.91962335      +0.57798785
9  -26414.63377681      +0.28584654
10  -26414.40491948      +0.22885733
11  -26414.21239776      +0.19252172
12  -26414.06799768      +0.14440008
13  -26413.97276037      +0.09523731
14  -26413.91491866      +0.05784171
15  -26413.88131232      +0.03360634
16  -26413.86234836      +0.01896395
17  -26413.85188992      +0.01045844
18  -26413.84622692      +0.00566300
已训练标签 打开 阀门 的HMM-GMM模型
1  -35731.84306326             +nan
2  -32452.06952634   +3279.77353692
3  -32248.27256356    +203.79696278
4  -32206.83322810     +41.43933546
5  -32202.41597890      +4.41724920
6  -32200.54539946      +1.87057944
7  -32199.42609988      +1.11929958
8  -32198.65863229      +0.76746759
9  -32198.13963593      +0.51899636
10  -32197.81556047      +0.32407547
11  -32197.62080942      +0.19475105
12  -32197.50544413      +0.11536529
13  -32197.43855672      +0.06688741
14  -32197.40068660      +0.03787012
15  -32197.37967787      +0.02100873
16  -32197.36820725      +0.01147061
17  -32197.36201904      +0.00618822
已训练标签 开启 风扇 五 秒 的HMM-GMM模型
1  -33616.90676674             +nan
```

```
2  -29800.80294488   +3816.10382186
3  -29019.58853749    +781.21440740
4  -28870.35281762    +149.23571986
5  -28861.20094997      +9.15186765
6  -28860.62359597      +0.57735400
7  -28859.95964504      +0.66395093
8  -28858.16997664      +1.78966840
9  -28855.14501717      +3.02495947
10  -28852.33809897      +2.80691820
11  -28850.06608781      +2.27201116
12  -28848.40257100      +1.66351681
13  -28847.60535341      +0.79721759
14  -28847.29625147      +0.30910194
15  -28847.11121256      +0.18503890
16  -28846.97205009      +0.13916247
17  -28846.90308930      +0.06896079
18  -28846.88449055      +0.01859875
19  -28846.88134933      +0.00314122
已训练标签 开启 阀门 的HMM-GMM模型
1  -41640.83460253             +nan
2  -38235.52974372   +3405.30485881
3  -35917.76448177   +2317.76526194
4  -35660.63021625    +257.13426553
5  -35649.80561875     +10.82459750
6  -35647.98445771      +1.82116104
7  -35647.38253258      +0.60192513
8  -35647.15010581      +0.23242677
9  -35647.02152332      +0.12858249
10  -35646.87457338      +0.14694994
11  -35646.33666762      +0.53790576
12  -35643.55446375      +2.78220388
13  -35641.16793480      +2.38652895
14  -35640.86871171      +0.29922309
15  -35640.82382662      +0.04488508
16  -35640.79944493      +0.02438169
17  -35640.78568455      +0.01376038
18  -35640.77912582      +0.00655874
已训练标签 开启 风扇 九 分钟 的HMM-GMM模型
1  -42722.30875219             +nan
2  -37534.26668437   +5188.04206781
3  -36580.17464325    +954.09204113
4  -36410.52180736    +169.65283588
5  -36406.38809338      +4.13371398
6  -36405.72097250      +0.66712088
7  -36405.53836802      +0.18260448
8  -36405.46349207      +0.07487595
9  -36405.43071727      +0.03277480
10  -36405.41711689      +0.01360037
11  -36405.41187238      +0.00524451
已训练标签 关掉 风扇 三 秒 的HMM-GMM模型
1  -33969.16057008             +nan
2  -30501.97639439   +3467.18417568
3  -29881.11362285    +620.86277154
4  -29790.80803243     +90.30559043
5  -29787.45647816      +3.35155427
6  -29785.00592262      +2.45055554
7  -29783.87078135      +1.13514127
8  -29783.46497699      +0.40580436
9  -29783.32483878      +0.14013820
10  -29783.27553660      +0.04930218
11  -29783.25796176      +0.01757484
12  -29783.25165385      +0.00630791
已训练标签 关 风扇 四 秒 的HMM-GMM模型
1  -45177.27555977             +nan
2  -39588.17157755   +5589.10398222
```

```
3  -38434.27749706   +1153.89408049
4  -38051.29984163    +382.97765543
5  -37978.11949275     +73.18034888
6  -37969.58710140      +8.53239135
7  -37967.77896457      +1.80813683
8  -37966.87190207      +0.90706250
.......
```

Problem 2

Maximum Likelihood Estimation (MLE) for Multivariate Gaussian Mean

The probability density function (PDF) for a multivariate Gaussian distribution is given by:

$$p(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

where:

- $\mathbf{x}$ is a d-dimensional random vector,
- μ is the d-dimensional mean vector,
- Σ is the $d \times d$ covariance matrix,
- $|\Sigma|$ is the determinant of the covariance matrix,
- Σ^{-1} is the inverse of the covariance matrix.

Likelihood Function

Given a set of n sampled data points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, the likelihood function $L(\mu, \Sigma)$ is the product of the individual probabilities:

$$L(\mu, \Sigma) = \prod_{i=1}^{n} p(\mathbf{x}_i|\mu, \Sigma)$$

Taking the logarithm of the likelihood function (log-likelihood), we get:

$$\log L(\mu, \Sigma) = \sum_{i=1}^{n} \log p(\mathbf{x}_i|\mu, \Sigma)$$

Substituting the expression for the multivariate Gaussian PDF:

$$\log L(\mu, \Sigma) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i-\mu)^T\Sigma^{-1}(\mathbf{x}_i-\mu)$$

Maximizing the Log-Likelihood

To find the MLE for μ, we take the derivative of the log-likelihood with respect to μ and set it equal to zero. The derivative with respect to μ is:

$$\frac{\partial}{\partial \mu} \log L(\mu, \Sigma) = \sum_{i=1}^{n} \Sigma^{-1}(\mathbf{x}_i - \mu)$$

Setting this equal to zero:

$$\sum_{i=1}^{n} (\mathbf{x}_i - \mu) = 0$$

Solving for μ, we obtain the Maximum Likelihood estimate for the mean:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$

Conclusion

The Maximum Likelihood estimate for the mean μ of a multivariate Gaussian distribution is simply the sample mean of the data points:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$