

Speech_Recognition_2022

Tongji University · Class of 2022 · School of Computer Science and Technology · Software Engineering · Machine Intelligence
Direction · Speech Recognition Coursework

Teacher: Ying Shen

Semester of instruction: 2024-2025, autumn semester

Task: GMM-HMM

1. Refer to the lab manual and submit a lab report while using the features extracted from your first assignment as input and observing the recognition results.
2. Estimation of mean parameters in multivariate Gaussian models using maximum likelihood estimation methods μ . Given a set of sampling data $X = \{x_1, x_2, \dots, x_n\}$.

Probability density functions for multivariate Gaussian models:

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

Parameter description

- D : Dimensions of the data
- $|\Sigma|$: covariance matrix Σ determinant of a function

GMM-HMM

In the experiment, students logged into the Huawei Cloud website, ran the GMM-HMM based continuous word speech recognition code on the MindSpore platform and given dataset provided by Huawei and recognised the textual content in the test audio.

Experimental Steps

Environment Setup

1. **Access Huawei Cloud ModelArts Console:** Log into your Huawei Cloud account, visit the Huawei Cloud website, and access the ModelArts console.
2. **Create a Notebook Training Job:** In the console, select "Notebook" > "Create Notebook" > Configure the training job and select the appropriate computing resources (here, choose the basic 2-core + 8GB setup).
3. **Start the Notebook and Enter Development Environment:** After creating the notebook, start it and enter the development environment where the subsequent code training and testing will be executed.

数据准备

1. **Upload Data to Server:**
 - Upload the `datas.zip` file to the server.
 - Open the terminal and run: `unzip datas.zip`.
2. **Install Required Python Libraries:**

```
pip install python_speech_features
pip install hmmlearn
```

ialized MindSpore | Idle CPU: 50% | MEM: 0.75/8 GB | EVS: 0.23/5 GB | NET: ↑ 1.89KB/s / ↓ 11.34KB/s Mode: Edit Ln 4, Col 31 Untitled.ip

- The screenshot displays the JupyterLab environment. On the left, the file explorer shows a directory structure with files like `__MACOSX`, `.modelarts`, `datas`, `lost+found`, `datas.zip`, and `hmm_gmm...`. The file `Untitled.ipynb` is selected. The main area shows a code editor with a Python script. The script includes imports for `os`, `pickle`, `numpy`, `scipy.io.wavfile`, `wav`, `python_speech_features`, `mfcc`, `hmmlearn`, `GMMHMM`, and `heapq`. It defines paths for training data, labels, test data, and the model. The `wav2mfcc` function processes audio files and extracts MFCC features. The `obtain_config` function is also defined.

```
40 ms [1] 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

40 ms [2] # 训练数据路径
train_data_path = os.path.join(os.getcwd(), "datas/train/speech" )
label_path = os.path.join(os.getcwd(), "datas/labels/trainprompts_m" )
# 测试数据路径
test_data_path = os.path.join(os.getcwd(), "datas/test/speech" )
# 模型保存路径
model_path = "hmm_gmm_model.pkl"

40 ms [3] def wav2mfcc(labels, data_paths):

    trng_data = {}
    for label, data_path in zip(labels, data_paths):
        # if label in trng_data.keys():
        #     print(label)
        write_pickle = True
        mfccs = []
        rate, sig = wvf.read(data_path)
        mfcc_feat = mfcc(sig, rate)
        mfccs.append(mfcc_feat)
        trng_data[label] = mfccs

    return trng_data

40 ms [4] def obtain_config(labels):
    conf = {}
    for label in labels:
```

1. Create the GMM-HMM Model:

- ### 1. Create the GMM-HMM Model:

The screenshot shows a Jupyter Notebook with the following code:

```
def obtain_config(labels):
    conf = {}
    for label in labels:
        conf[label] = {}
        conf[label]["n_components"] = 2
        conf[label]["n_mix"] = 2
    return conf

def get_hmm_gmm(trng_data=None, GMM_configs=None, model_path="hmm_gmm_model.pkl", from_file=False):
    hmm_gmm = {}
    if not from_file:
        for label, trng_data in trng_data.items():
            GMM_config = GMM_configs[label]

            hmm_gmm[label] = GMMGMM(
                n_components=GMM_config["n_components"],
                n_mix=GMM_config["n_mix"])

            if trng_data:
                hmm_gmm[label].fit(np.vstack(trng_data))

    pickle.dump(hmm_gmm, open(model_path, "wb"))
    else:
        hmm_gmm = pickle.load(open(model_path, "rb"))
    return hmm_gmm

def train(train_data_path, label_path, model_path):
    with open(os.path.join(label_path)) as f:
        labels = f.readlines() # 读取所有行，返回列表
    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
```

2. Read Training Data and Train the Model:

The screenshot shows a Jupyter Notebook with the following code:

```
def train(train_data_path, label_path, model_path):
    with open(os.path.join(label_path)) as f:
        labels = f.readlines() # 读取所有行，返回列表
    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

hmm_gmm = train(train_data_path, label_path, model_path)
print(hmm_gmm)

{'开灯七小时': GMMGMM(algorithm='viterbi', covariance_type='diag',
    covars_prior=array([[[-1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5,
    -1.5, -1.5, -1.5],
    [-1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5],
    [-1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5],
    [-1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5]]],
    covars...
    means_prior=array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]],
    means_weight=array([[0., 0.],
    [0., 0.]]), min_covar=0.001,
    n_components=2, n_iter=10, n_mix=2, params='stmcw', random_state=None,
    startprob_prior=1.0, tol=0.01, transmat_prior=1.0, verbose=False,
    weights_prior=array([[1., 1.],
    [1., 1.])), '关闭 风扇': GMMGMM(algorithm='viterbi', covariance_type='diag',
    covars_prior=array([[[-1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5,
    -1.5, -1.5, -1.5],
    [-1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5],
    [-1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5],
    [-1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5, -1.5]]],
    covars...
    means_prior=array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]],
    means_weight=array([[0., 0.],
    [0., 0.]]), min_covar=0.001,
    n_components=2, n_iter=10, n_mix=2, params='stmcw', random_state=None,
    startprob_prior=1.0, tol=0.01, transmat_prior=1.0, verbose=False,
    weights_prior=array([[1., 1.],
    [1., 1.])))
```

3. Test the Model:

The screenshot shows a Jupyter Notebook with the following code:

```
def test_file(test_file, hmm_gmm):
    rate, sig = wavf.read(test_file)
    mfcc_feat = mfcc(sig, rate)
    pred = {}
    for model in hmm_gmm:
        pred[model] = hmm_gmm[model].score(mfcc_feat)
    return get_nbest(pred, 2), pred

# 获取topN的结果
def get_nbest(d, n):
    return heapq.nlargest(n, d, key=lambda k: d[k])

def predict_label(file, hmm_gmm):
    predicted = test_file(file, hmm_gmm)
    return predicted

wave_path = os.path.join(test_data_path, "T0001.wav")
# wave_path = os.path.join(train_data_path, "S0001.wav")
predicted, probs = predict_label(wave_path, hmm_gmm)
print("PREDICTED: %s" % predicted[0])

PREDICTED: 关掉 风扇 二秒
```

The complete code and output are in the **GMM-HMM.ipynb** file, and the feature values generated by the library functions are saved in the **hmm_gmm_model.pkl** file.

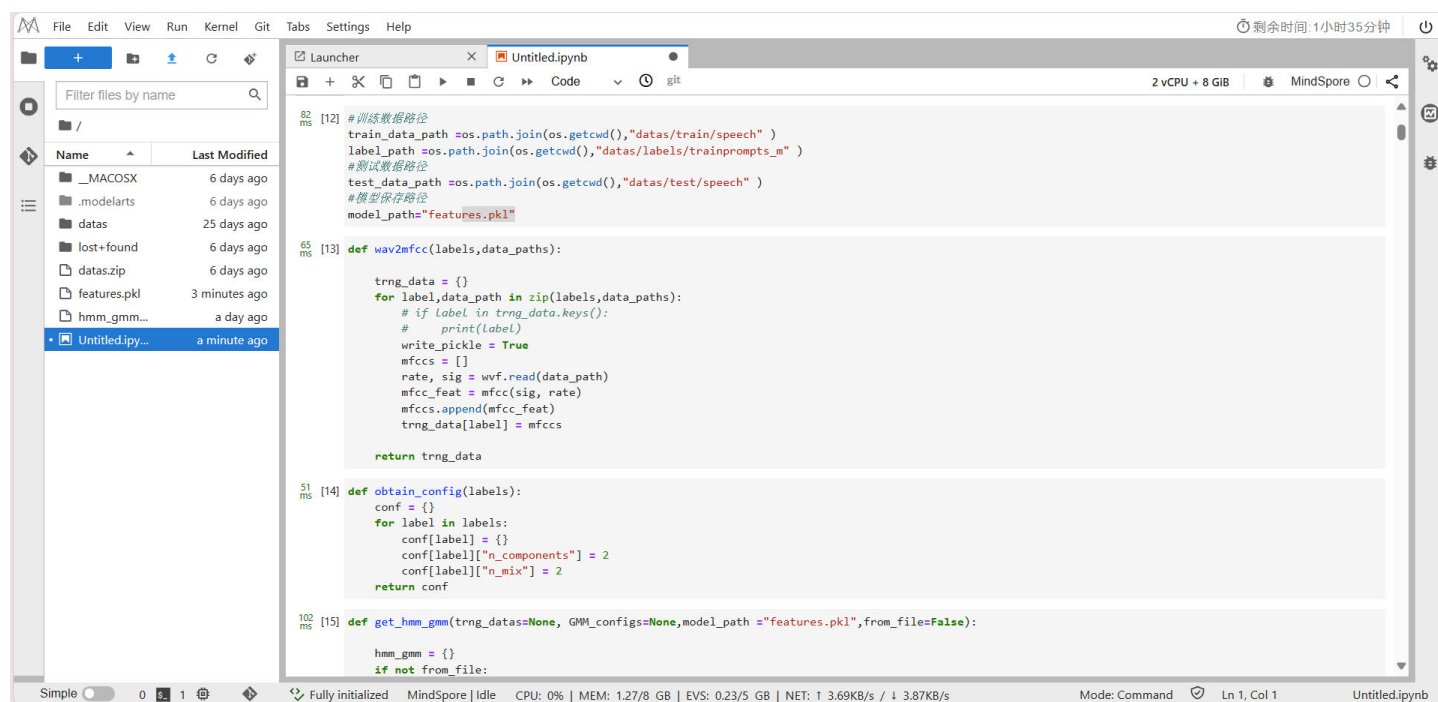
Using Features Extracted from Your First Assignment as Input

Experimental Observations

The features extracted from your first assignment are saved in the **feature.pkl** file, and the detailed code is in the **MFCC_Extraction.ipynb** file.

The extracted features include:

- The MFCC features of the audio.
- The standardized and processed feature vectors.



```
File Edit View Run Kernel Git Tabs Settings Help
Launcher x Untitled.ipynb
2 vCPU + 8 GiB MindSpore
[12] #训练数据路径
train_data_path = os.path.join(os.getcwd(), "datas/train/speech" )
label_path = os.path.join(os.getcwd(), "datas/labels/trainprompts_m" )
#测试数据路径
test_data_path = os.path.join(os.getcwd(), "datas/test/speech" )
#模型保存路径
model_path = "features.pkl"

[13] def wav2mfcc(labels, data_paths):

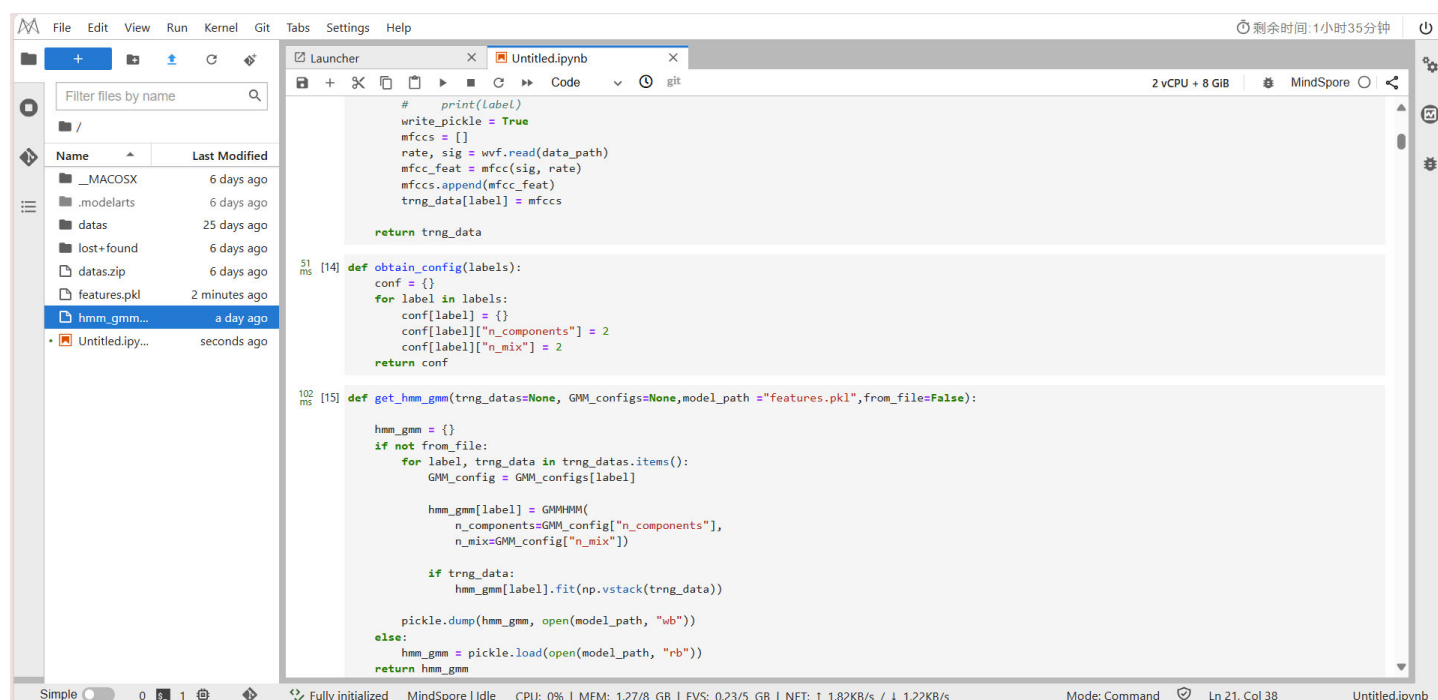
    trng_data = {}
    for label, data_path in zip(labels, data_paths):
        # if label in trng_data.keys():
        #     print(label)
        write_pickle = True
        mfccs = []
        rate, sig = wvf.read(data_path)
        mfcc_feat = mfcc(sig, rate)
        mfccs.append(mfcc_feat)
        trng_data[label] = mfccs

    return trng_data

[14] def obtain_config(labels):
    conf = {}
    for label in labels:
        conf[label] = {}
        conf[label]["n_components"] = 2
        conf[label]["n_mix"] = 2
    return conf

[15] def get_hmm_gmm(trng_datas=None, GMM_configs=None, model_path = "features.pkl", from_file=False):

    hmm_gmm = {}
    if not from_file:
```



```
File Edit View Run Kernel Git Tabs Settings Help
Launcher x Untitled.ipynb
2 vCPU + 8 GiB MindSpore
#     print(label)
write_pickle = True
mfccs = []
rate, sig = wvf.read(data_path)
mfcc_feat = mfcc(sig, rate)
mfccs.append(mfcc_feat)
trng_data[label] = mfccs

return trng_data

[14] def obtain_config(labels):
    conf = {}
    for label in labels:
        conf[label] = {}
        conf[label]["n_components"] = 2
        conf[label]["n_mix"] = 2
    return conf

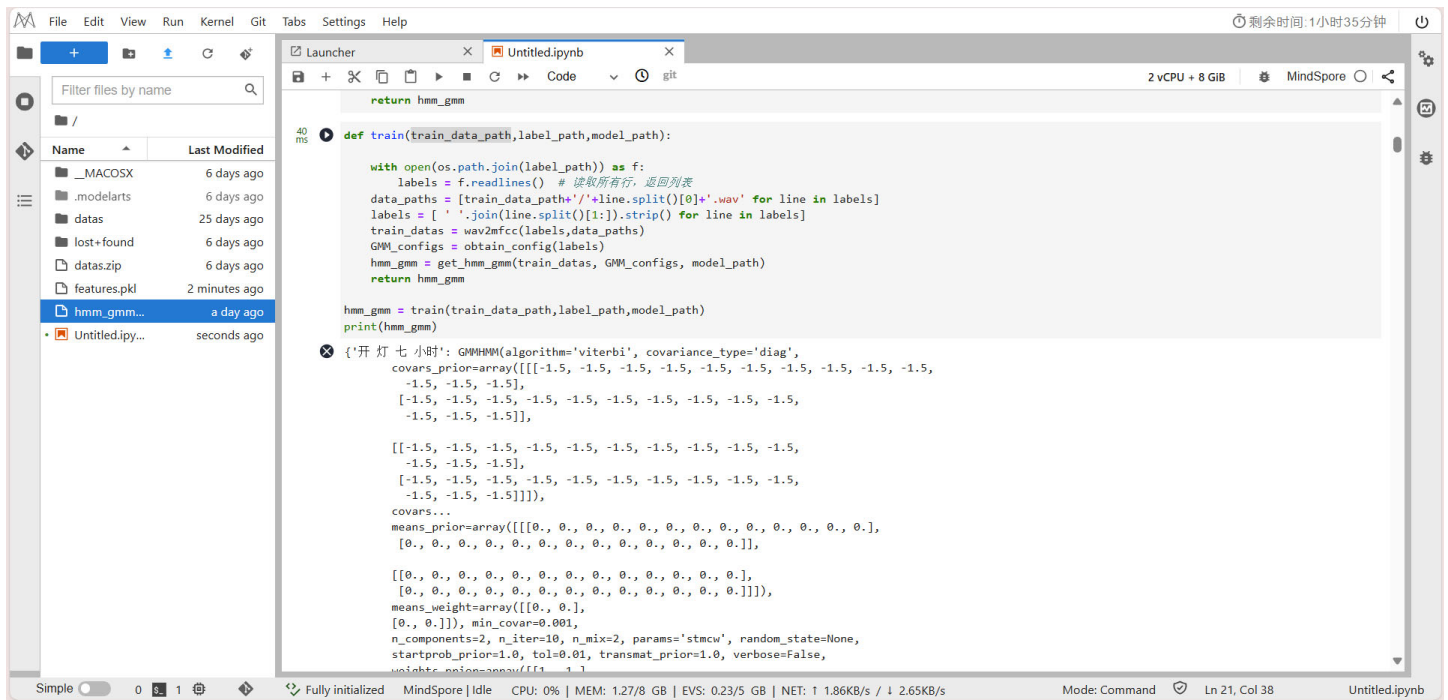
[15] def get_hmm_gmm(trng_datas=None, GMM_configs=None, model_path = "features.pkl", from_file=False):

    hmm_gmm = {}
    if not from_file:
        for label, trng_data in trng_datas.items():
            GMM_config = GMM_configs[label]

            hmm_gmm[label] = GMMHMM(
                n_components=GMM_config["n_components"],
                n_mix=GMM_config["n_mix"])

            if trng_data:
                hmm_gmm[label].fit(np.vstack(trng_data))

            pickle.dump(hmm_gmm, open(model_path, "wb"))
        else:
            hmm_gmm = pickle.load(open(model_path, "rb"))
    return hmm_gmm
```

The complete code and output are in the **mineGMM-HMM.ipynb** file.

Summary

Maximum Likelihood Estimation Steps

1. Likelihood Function

Suppose we have n sample data points x_1, x_2, \dots, x_n , the joint likelihood function is the product of the probability density function of each data point:

$$L(\mu|X) = \prod_{i=1}^n p(x_i|\mu, \Sigma)$$

2. Log-Likelihood Function

To simplify the calculation, we usually take the logarithm of the likelihood function to get the log-likelihood function:

$$\log L(\mu|X) = \sum_{i=1}^n \log p(x_i|\mu, \Sigma)$$

Substituting the probability density function of a multivariate Gaussian distribution:

$$\log L(\mu|X) = -\frac{n}{2} \log(2\pi) - \frac{n}{2} \log |\Sigma| - \frac{1}{2} \sum_{i=1}^n (x_i - \mu)^\top \Sigma^{-1} (x_i - \mu)$$

3. Derivative with Respect to μ

To maximize the log-likelihood function, we take the derivative with respect to μ and set it to zero:

$$\frac{\partial}{\partial \mu} \log L(\mu|X) = \frac{\partial}{\partial \mu} \left(-\frac{1}{2} \sum_{i=1}^n (x_i - \mu)^\top \Sigma^{-1} (x_i - \mu) \right)$$

Since Σ is a constant matrix, after differentiating, we get:

$$\frac{\partial}{\partial \mu} \log L(\mu|X) = \Sigma^{-1} \sum_{i=1}^n (x_i - \mu)$$

Setting the derivative equal to zero, we solve for μ :

$$\sum_{i=1}^n (x_i - \mu) = 0$$

Thus:

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

4. Final Result

Therefore, the maximum likelihood estimate for the mean μ is the sample mean:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i$$