《语音识别》
-语音信号处理

目录

1 实验介绍

本章实验主要介绍了语音信号处理的相关知识。

1.1 实验目的

本章实验的主要目的是掌握语音处理相关知识点，了解语音处理相关基础知识，了解语音信号分析的一般流程，掌握 MFCC 提取过程。

1.2 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

实验	简述	难度	软件环境	开发环境
音频录制	使用 Python 工具 pyaudio 完成音频录制	中级	Python3.7	PC 64bit ModelArts
语音信号预处理	使用 Python 及 HTK 工具完成语音信号处理过程。	中级	Python3.7	PC 64bit ModelArts
语音信号分析	通过快速傅里叶变换完成语音信号的分析。	中级	Python3.7	PC 64bit ModelArts

1.3 开发平台介绍

2 音频录制

2.1 实验介绍

通过 Python 工具库 pyaudio，调用本地麦克风阵列实现音频的录制。

2.2 实验环境要求

Python3.7

PyAudio

2.3 实验过程

2.3.1 音频录制

```
import wave
from pyaudio import PyAudio,paInt16

framerate=16000    #
NUM_SAMPLES=2000          #
channels=1# 声道
sampwidth=2   #
TIME=5     #

def save_wave_file(filename,data):
    with wave.open(filename,'wb') as wf: #
        wf.setnchannels(channels) #
        wf.setsampwidth(sampwidth) #
        wf.setframerate(framerate) #
        wf.writeframes(b"".join(data))

def my_record(filename):
```

```
    pa=PyAudio()
    stream=pa.open(format = paInt16,channels=1, #
                        rate=framerate,input=True,
                        frames_per_buffer=NUM_SAMPLES)
    my_buf=[]
    count=0
    while count<TIME*8:#
        string_audio_data = stream.read(NUM_SAMPLES)
        my_buf.append(string_audio_data)
        count+=1
        print('.',end="")
    save_wave_file(filename+".wav",my_buf)
    stream.close()
my_record("test")
```

2.4 实验总结

本小结实验主要通过 pyaudio 实现音频文件的录制。

3 语音信号预处理实验

3.1 实验介绍

本章的主要内容就是基于 Python3.7 和框架 HTKFeat 和 HTK 进行简单的语音数据的预处理；经过实验之后，让学员掌握对音频数据的处理操作。

3.2 实验环境要求

1、 python3.7

3.3 实验过程

3.3.1 实验准备

步骤 1 导入实验所需模块

```
import warnings
warnings.filterwarnings('ignore')
import os
import sys
sys.path.append(os.path.join(os.getcwd(),"asr\\data1\\"))
from HTKFeat import MFCC_HTK #
from HTK import HCopy,HTKFile
import numpy as np
import matplotlib.pyplot as plt
```

步骤 2 设置数据路径

```
# 配置数据路径
data_path =os.path.join(os.getcwd(),'asr/data1/test.wav')
```

3.3.2 加载数据并显示频谱

```
mfcc=MFCC_HTK()
```

```
signal = mfcc.load_raw_signal(data_path)
signal = signal[100:]
sig_len=signal.size/16000 #in seconds

plt.figure(figsize=(15,4))
t=np.linspace(0,sig_len,signal.size)
plt.plot(t,signal)
plt.figure(figsize=(15,4))
s=plt.specgram(signal,Fs=16000)
plt.xlim(0,sig_len)
```

输出：

3.3.3 音频处理

步骤 1 移除数据中心值(即均值)

```
print( "Before: "+str(np.mean(signal)))

signal = signal - np.mean(signal)

print( "After: "+str(np.mean(signal)))
```

输出：

```
Before: 0.43412656707454395
After: -2.4549222183486864e-14
```

步骤 2 分帧

```
win_shift=160
win_len=400


sig_len=len(signal)

win_num=np.floor((sig_len-win_len)/win_shift).astype('int')+1

wins=[]
for w in range(win_num):



    s=w*win_shift
    e=s+win_len
```

```
    win=signal[s:e].copy()

    wins.append(win)

wins=np.asarray(wins)
```

步骤 3 预加重

```
k=0.97
h = [1,-k]
f=np.linspace(0,8000,257)
plt.plot(f,np.abs(np.fft.rfft(h,n=512)))
plt.xlabel('Frequency')
plt.ylabel('Amplitude correction')
```

输出:

```
for win in wins:
    win-=np.hstack((win[0],win[:-1]))*k
plt.plot(np.abs(np.fft.rfft(h,n=328)),wins)
plt.xlabel('Amplitude correction')
plt.ylabel('Frequency')
```

输出:

```
Text(0, 0.5, 'Frequency')
```

步骤 4 加窗

```
rect = np.ones(400)

plt.figure(figsize=(12,4))
plt.subplot(2,1,1)
plt.stem(rect)
plt.xlim(-100,500)
plt.ylim(-0.1,1.1)
plt.title('Square window')

hamm = np.hamming(400)
plt.figure(figsize=(12,4))
plt.subplot(2,1,2)
plt.stem(hamm)
plt.xlim(-100,500)
plt.ylim(-0.1,1.1)
plt.title('Hamming function')
```

输出：

```
for win in wins:
    win*=hamm
plt.plot(np.abs(np.log(np.abs(np.fft.rfft(hamm,n=328)))),wins)
plt.xlabel('Frequency')
plt.ylabel('Amplitude (log)')
```

输出：

3.4 实验总结

本章基于 Python 实现了音频数据信号的展示。

4 语音信号分析

4.1 实验介绍

本章节主要是对语音数据进行快速傅里叶变换和 MFCC 系数的提取，完成对于语音信号的分析。

4.2 实验环境要求

Python3,7

4.3 实验过程

4.3.1 快速傅里叶变换

```
import numpy as np
import os
import sys
import matplotlib.pyplot as plt
sys.path.append(os.path.join(os.getcwd(),"asr\\data1\\"))
from HTKFeat import MFCC_HTK
from HTK import HCopy,HTKFile

# 配置数据路径
data_path =os.path.join(os.getcwd(),'asr/data1/test.wav')
mfcc=MFCC_HTK()
win_shift=160
win_len=400
signal = mfcc.load_raw_signal(data_path)
signal = signal[100:]

sig_len=len(signal)

win_num=np.floor((sig_len-win_len)/win_shift).astype('int')+1

wins=[]
for w in range(win_num):

    #t 每个窗的开始和结束
    s=w*win_shift
    e=s+win_len

    win=signal[s:e].copy()

    wins.append(win)

wins=np.asarray(wins)
fft_len=(2**(np.floor(np.log2(win_len))+1)).astype('int')
```

```
ffts=[]
for win in wins:
    win=np.abs(np.fft.rfft(win,n=fft_len)[:-1])
    ffts.append(win)

ffts=np.asarray(ffts)

plt.figure(figsize=(10,5))
plt.pcolormesh(ffts.T,cmap='gray')
plt.xlim(0,win_num)
plt.ylim(0,fft_len/2)
```

输出：

4.3.2 梅尔频谱倒频系数

```
freq2mel = lambda freq: 1127*(np.log(1+((freq)/700.0)))

f = np.linspace(0,8000,1000)
m = freq2mel(f)

plt.plot(f,m)
plt.xlabel('Frequency')
plt.ylabel('Mel')
```

输出：

4.3.3 滤波器

步骤 1 三角滤波器

```
mfcc.create_filter(26)

plt.figure(figsize=(15,3))
for f in mfcc.filter_mat.T:
    plt.plot(f)
plt.xlim(0,256)
```

输出：

步骤 2 快速傅里叶变换的结果与滤波器乘积

```
melspec=[]
for f in ffts:
    m = np.dot(f,mfcc.filter_mat)
    melspec.append(m)
melspec=np.asarray(melspec)

plt.figure(figsize=(15,5))
plt.pcolormesh(melspec.T,cmap='gray')
plt.xlim(0,win_num)
plt.ylim(0,26)
```

输出：

步骤 3 放大数据中的细节同时衰减高光

```
mels = np.log(melspec)

plt.figure(figsize=(15,5))
plt.pcolormesh(mels.T,cmap='gray')
plt.xlim(0,win_num)
plt.ylim(0,26)
```

输出：

4.3.4 离散余弦变换

步骤 1 离散余弦变换效果

```
filter_num=26
mfcc_num=12

dct_base=np.zeros((filter_num,mfcc_num));
for m in range(mfcc_num):
    dct_base[:,m]=np.cos((m+1)*np.pi/filter_num*(np.arange(filter_num)+0.5))

plt.figure(figsize=(6,3))
plt.pcolormesh(dct_base.T,cmap='gray')
plt.xlim(0,24)
plt.ylim(0,12)
plt.xlabel('Filters')
plt.ylabel('MFCCs')
```

输出：

步骤 2 获得 MFCC

```
#将梅尔频谱图与 DCT 矩阵进行矩阵乘法以获得 MFCC
filter_num=26
mfcc_num=12

mfccs=[]
for m in mels:
    c=np.dot(m,dct_base)
    mfccs.append(c)
mfccs=np.asarray(mfccs)

plt.figure(figsize=(15,5))
plt.pcolormesh(mfccs.T,cmap='RdBu')#cmap:RdBu,Oranges,YlGn,OrRd,gray，...
plt.xlim(0,win_num)
plt.ylim(0,mfcc_num)
```

输出：

步骤 3 将 MFCC 归一化

```
mfnorm = np.sqrt(2.0 / filter_num)
mfccs*=mfnorm
```

4.3.5 对比倒频谱能量与原始能量

```
raw_energy=[]
for win in wins:
    raw_energy.append(np.log(np.sum(win**2)))
raw_energy=np.asarray(raw_energy)

ceps_energy=[]
for m in mels:
    ceps_energy.append(np.sum(m)*mfnorm)
ceps_energy=np.asarray(ceps_energy)

plt.figure(figsize=(15,5))
plt.plot(raw_energy,'r',label='Raw energy')
plt.plot(ceps_energy,'b',label='Cepstral energy')#倒频谱能量
plt.xlim(0,win_num)
plt.legend()
```

输出：

4.4 实验总结

本章的主要基于框架 HTKFeat 和 HTK 进行简单的语音数据的预处理操作。