真伪语音鉴别实验报告

分 营： 智能语音计算

学 校： 大连理工大学

专 业： 软件工程

姓 名： 陈依苓

日 期： 2021.7.18

目录

[1 真伪语音鉴别 2](#_Toc77518374)

[1.1 实验介绍 2](#_Toc77518375)

[1.1.1 实验目的 2](#_Toc77518376)

[1.1.2 实验环境 2](#_Toc77518377)

[1.2 实验步骤 2](#_Toc77518378)

[1.2.1 语音识别工程创建 2](#_Toc77518379)

[1.2.2 语音预处理 3](#_Toc77518380)

[1.2.3 GMM模型 4](#_Toc77518381)

[1.2.4 语音识别 5](#_Toc77518382)

[1.3 参数试凑 6](#_Toc77518383)

[1.3.1 MFCC特征提取 6](#_Toc77518384)

[1.3.2 GMM模型 6](#_Toc77518385)

# 真伪语音鉴别

## 实验介绍

### 实验目的

理解语音预处理的基本操作。

理解语音识别的基本原理。

掌握语音识别的基本开发流程。

### 实验环境

Windows10

Python3.8

Jupyter nodebook

## 实验步骤

### 语音识别工程创建

工具包安装

安装本次实验中所需要的工具包

pip install spafe==0.1.2

pip install scipy==1.6.2

pip install numpy==1.19.5

pip install pickle

获取数据集

此次用到的数据是数据集中train文件夹下的数据，用作评估模型的数据集是dev文件夹下的数据

train\_data\_path = './audio/train/flac/'

train\_label\_path = './audio/train.txt'

signs = np.loadtxt(train\_label\_path,delimiter=' ',dtype=str)

### 语音预处理

定义参数

一些可能修改的参数

# 规定一些参数

NOR\_LEN = 32000 # 归一化后的文件长度

frame\_size = 0.025 # 帧长 25ms

frame\_stride = 0.01 # 帧移 10ms

NFFT = 512 # N点 FFT 短时傅里叶变换

nfilt = 40 # 梅尔滤波器的个数

num\_ceps = 15 # 倒谱系数

采样

以16KHz的采样率对实验数据进行采样。

归一化音频文件的长度

归一化音频长度为2s,32000数据点

def normalizeVoiceLen(y,normalizedLen): # y：data normalizedLen:32000

'''

音频文件归一化长度

'''

nframes=len(y)

y = np.reshape(y,[nframes,1]).T

#归一化音频长度为2s,32000数据点

if(nframes<normalizedLen): # 长度不足用0补

res=normalizedLen-nframes

res\_data=np.zeros([1,res],dtype=np.float32)

y=np.c\_[y,res\_data]

else:

y=y[:,0:normalizedLen]

return y[0] # 向量

MFCC特征提取

MFCC特征提取的步骤主要分为预加重、分帧、加窗、FFT、构建Mel滤波器组、对数运算、DCT等。这里直接使用spafe库中的mfcc函数对每条数据提取特征

def get\_mfcc\_features(files):

'''

MFCC特征提取

files flac文件路径

'''

mfcc\_features = np.asarray(())

for f in files:

data,sample\_rate = sf.read(f)

data = normalizeVoiceLen(data,NOR\_LEN)

features = mfcc(data,fs=sample\_rate,nfft=NFFT,num\_ceps=num\_ceps)

features = preprocessing.scale(features)

if mfcc\_features.size == 0:

mfcc\_features = features

else:

mfcc\_features=np.append(mfcc\_features,features,axis=0)

return mfcc\_features

### GMM模型

划分数据集

本实验需要分别为真语音和伪语音分别构建GMM模型，因此将训练集数据分成2个集合。

# 形成2个文件路径集合，用于生成2个GMM，spoof伪，bonafide真

files\_spoof = []

files\_bonafide = []

for s in signs:

if s[1]=='spoof':

files\_spoof.append(train\_data\_path+s[0]+'.flac')

else:

files\_bonafide.append(train\_data\_path+s[0]+'.flac')

print('spoof训练集大小',len(files\_spoof))

print('bonafide训练集大小',len(files\_bonafide))

x\_train\_spoof,x\_train\_bonafide = get\_mfcc\_features(files\_spoof),get\_mfcc\_features(files\_bonafide)

print('spoof:',x\_train\_spoof.shape,' bonafide:',x\_train\_bonafide.shape)

构建GMM模型并训练

GMM模型，即高斯混合模型，可以看作是由 K 个单高斯模型组合而成的模型，这 K 个子模型是混合模型的隐变量。使用EM算法进行参数的学习，用于含有隐变量（Hidden variable）的概率模型参数的最大似然估计。本实验直接使用sklearn库的GMM

# 模型训练

from sklearn.mixture import GaussianMixture as GMM

gmm = GMM(n\_components=8,covariance\_type='diag',n\_init=3)

gmm\_model\_spoof = gmm.fit(x\_train\_spoof)

gmm = GMM(n\_components=8,covariance\_type='diag',n\_init=3)

gmm\_model\_bonafide = gmm.fit(x\_train\_bonafide)

# print(gmm\_model\_spoof.converged\_,gmm\_model\_bonafide.converged\_)

模型保存

分别将训练好的模型写入spoof.gmm和bonafide.gmm文件中

# 写模型

picklefile\_spoof = 'spoof.gmm'

picklefile\_bonafide = 'bonafide.gmm'

pickle.dump(gmm\_model\_spoof,open('./'+picklefile\_spoof,'wb'))

pickle.dump(gmm\_model\_bonafide,open('./'+picklefile\_bonafide,'wb'))

### 语音识别

使用训练好的声学模型实现真伪语音鉴别。

加载模型

# 打开模型

modelpath = './'

gmm\_files=[os.path.join(modelpath,fname) for fname in os.listdir(modelpath) if fname.endswith('.gmm')]

models=[pickle.load(open(fname,'rb')) for fname in gmm\_files]

results = [fname[len(modelpath):-4] for fname in gmm\_files] # 标签下标到字符串的映射

for fname in gmm\_files:

print("fname:",fname)

print(results)

处理测试集

# 评估

eval1\_data\_path = './audio/eval1/flac/' # path to evaluating data

test\_label\_path = './audio/dev.txt'

files\_eval1=os.listdir(eval1\_data\_path)

with open('eval1.txt','w') as f:

for i,file in enumerate(files\_eval1):

file\_path = eval1\_data\_path+file

audio,sr=sf.read(file\_path)

feature=get\_Mfcc(sr,audio)

scores = None

# 2个模型，所以长度是2

log\_likelihood=np.zeros(len(models))

# 循环加载这两个模型

for k in range(len(models)):

gmm = models[k]

##求概率

scores=np.array(gmm.score(feature).reshape(1,-1))

log\_likelihood[k]=scores.sum()

winner=np.argmax(log\_likelihood) # 对应下标

# 预测结果写入txt

predict\_label = results[winner]

f.write(file[0:-5]+' '+predict\_label+'\n')

## 参数试凑

### MFCC特征提取

参数包括采样率、帧长、帧移、梅尔滤波器个数、倒谱系数、短时傅里叶变换时长。依据经验，各参数的取值如下表

MFCC特征提取参数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sample rate(Hz) | Frame size(ms) | Frame stride(ms) | NFFT | nfilt | Num\_ceps |
| 16000 | 0.025 | 0.01 | 512 | 40 | 13 |

### GMM模型

主要涉及高斯分量个数n\_component、协方差类别covariance\_type。

对于参数不同的取值，在验证集上的预测结果如下

GMM参数调试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n\_components | accuracy | precision\_spoof | precision\_bonafide | recall\_spoof | recall\_bonafide |
| 12 | 0.8472 | 0.9864 | 0.3936 | 0.8412 | 0.8992 |
| 11 | 0.844 | 0.9904 | 0.3921 | 0.8340 | 0.9302 |
| 10 | 0.8432 | 0.9894 | 0.3901 | 0.8340 | 0.9224 |
| 9 | 0.8436 | 0.9899 | 0.3911 | 0.8340 | 0.9263 |
| 8 | 0.8464 | 0.9889 | 0.395 | 0.8380 | 0.9186 |