期末项目报告

说明

该仓库是大数据案例事务课程的期末项目。项目内容为识别音频是男声还是女声。训练集包含2000个实例，测试集包含500个实例。

数据探索

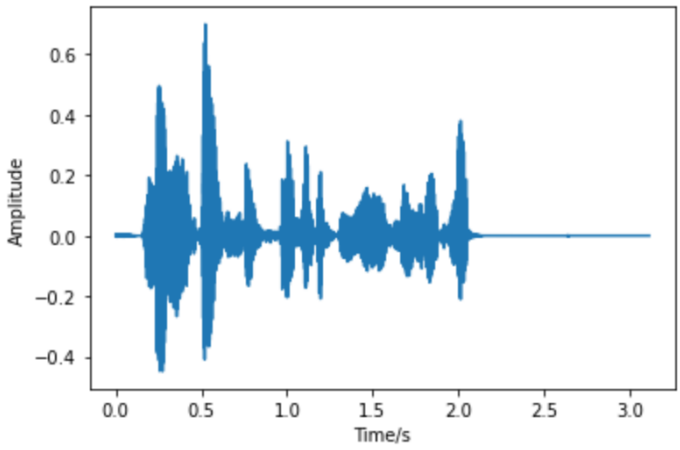
首先，我们需要对数据集进行初步探索，以对数据集有一个初步的认识，为后续的特征工程和模型构建做准备。（探索过程见exploration.ipynb）

数据格式

我们使用librosa库读取数据，采用率设为16000，读取后画出波形图

```
[5]
fs = 16000
sig, _ = librosa.load("data/train/1.wav", sr=fs)
time = np.arange(0, len(sig)) * (1.0/fs)
plt.plot(time, sig, color="C0")
plt.xlabel("Time/s")
plt.ylabel("Amplitude")
plt.show()
```

Python



读取后的信号数组为一维数组，长度根据音频长度改变。

```
[4]
sig.shape
```

✓ 0.5s Python

```
(49858,)
```

音频时长

我们想要查看各音频的时长分布，使用librosa.get_duration函数。

```python
times = []
for i in trange(1, 2001):
    y_ps, sr = librosa.load(f"data/train/{i}.wav")
    time = librosa.get_duration(y_ps, sr)
    times.append(time)
```

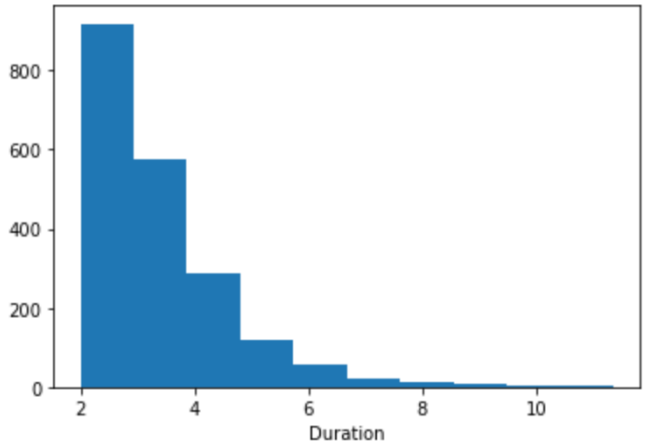
[7] Python

```
100%|██████████| 2000/2000 [02:01<00:00, 16.39it/s]
```

画出音频时长的直方图。

```python
plt.hist(times)
plt.xlabel("Duration")
plt.show()
```

[9] Python



可以看到，大部分音频在2-4s，最长的音频在11s左右。

平均基频

根据资料，男生和女生的基频有着显著不同，可以作为一个特征。男声的基音频率大都在100-200HZ之间，而女声则在200-350HZ之间，因此，我们尝试使用pysptk.sptk.swipe函数计算出各个音频的基频。

```python
df = pd.read_excel('data/train.xlsx')
lb = list(df['label'])
l = []
for i in trange(1, 2001):
    fs, data_audio = wav.read(f"data/train/{i}.wav")
    size_step = 0.02
    data_audiof = data_audio.astype(np.float64)
    size_stepS = size_step * fs
    bf = pysptk.sptk.swipe(data_audiof, fs, int(size_stepS), min=50, max
    q = bf[bf>0]
    if q.shape[0] > 0:
        l.append([f"{i}.wav",lb[i-1],q.mean()])
    else:
        l.append([f"{i}.wav",lb[i-1],0])
```

[23] Python

```
100%|██████████| 2000/2000 [02:31<00:00, 13.24it/s]
```

我们查看男生平均基频和女生平均基频的均值

```
z = pd.read_excel('data/train_preprocessed.xlsx')
s = z['bf'].groupby(z['label'])
print(s.mean())
```

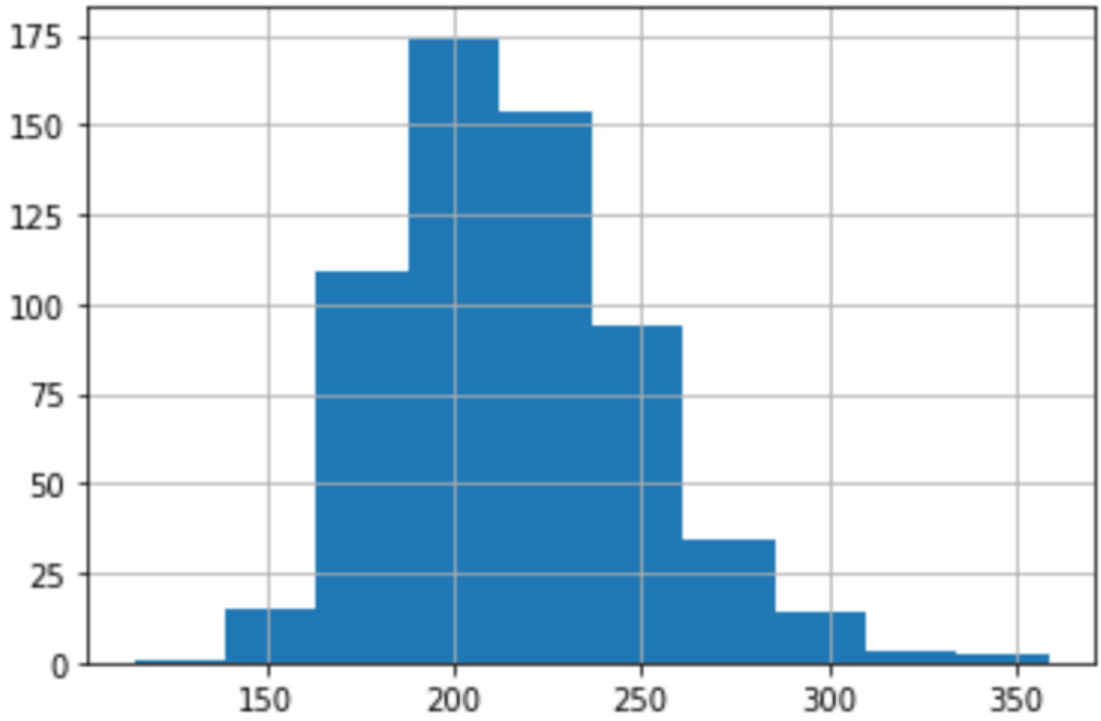
[30] Python

```
label
0    215.819949
1    143.669548
Name: bf, dtype: float64
```

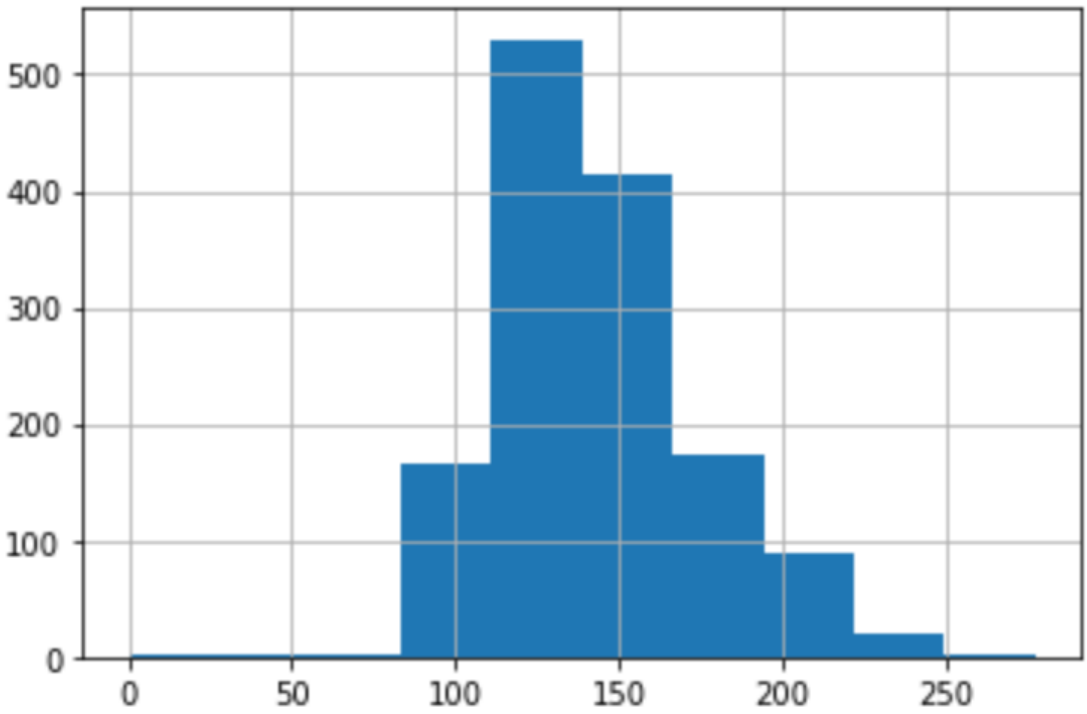
可以看到两者的差异较大，说明平均基频是一个区分度较大的特征。

我们还可以画出男生基频和女生基频的直方图。

女生基频为：

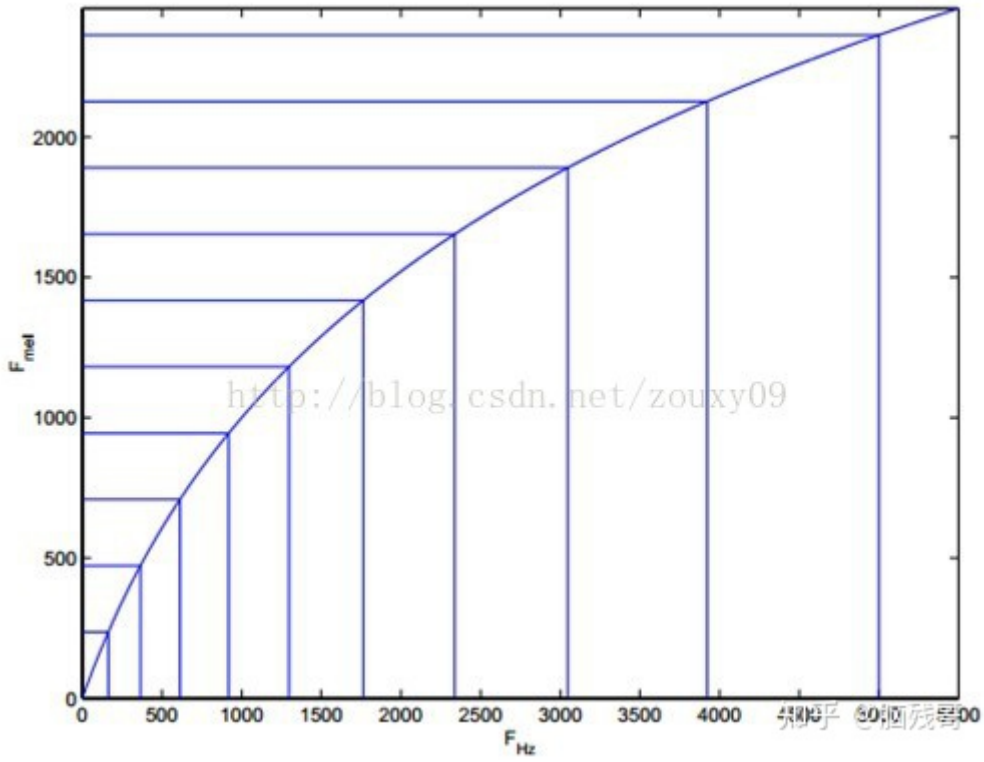


男生基频为：



梅尔频谱

研究表明，人类对频率的感知并不是线性的，并且对低频信号的感知要比高频信号敏感。梅尔频谱就是在梅尔刻度下的语谱图，是通过语谱图与若干个梅尔滤波器点乘得到的图谱。梅尔刻度是频率的对数刻度：



获得梅尔频谱图的具体步骤为：

1. 获取音频的信号数据
2. 对音频信号进行快速傅里叶变换，获得频谱
3. 通过对信号的多个窗口执行快速傅里叶变换得到语谱图
4. 对语谱图将频率刻度转换为梅尔刻度，通过梅尔滤波器变换为梅尔频谱

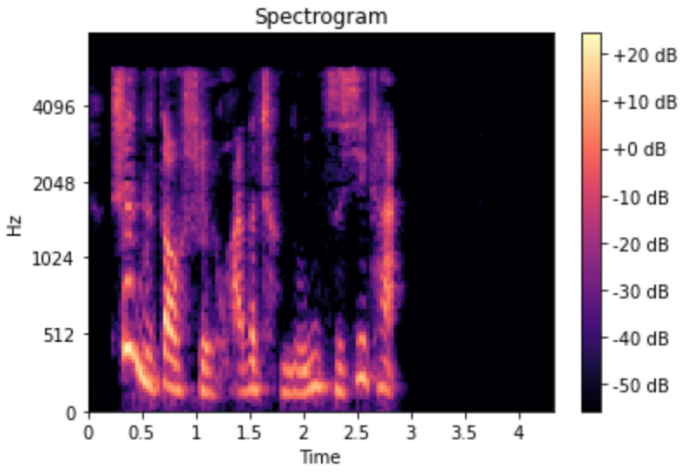
我们可以用librosa.feature.melspectrogram得到梅尔频谱，我们分别计算出男生的梅尔频谱和女生的梅尔频谱。

```
# 女生的梅尔频谱
spec = wav2melspectrogram("data/train/1.wav")
```

✓ 0.9s Python

```
librosa.display.specshow(spec, y_axis='mel', sr=fs, fmax=8000, x_axis='time')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram')
plt.show()
```

✓ 0.3s Python

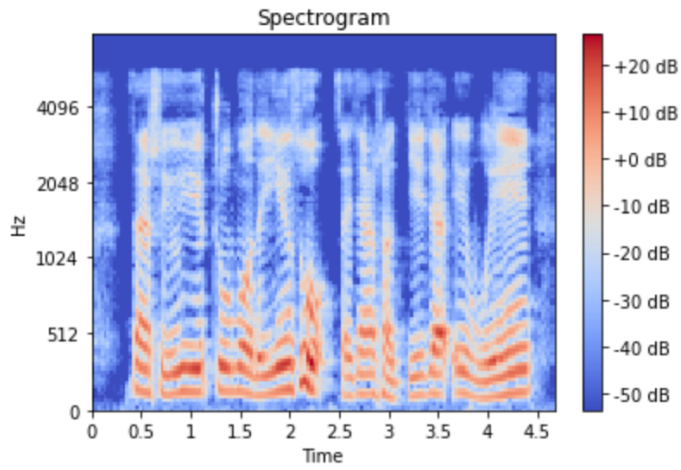



```
# 男生的梅尔频谱
spec = wav2melspectrogram("data/train/4.wav")
```

✓ 0.9s Python

```
librosa.display.specshow(spec, y_axis='mel', sr=fs, fmax=8000, x_axis='time')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram')
plt.show()
```

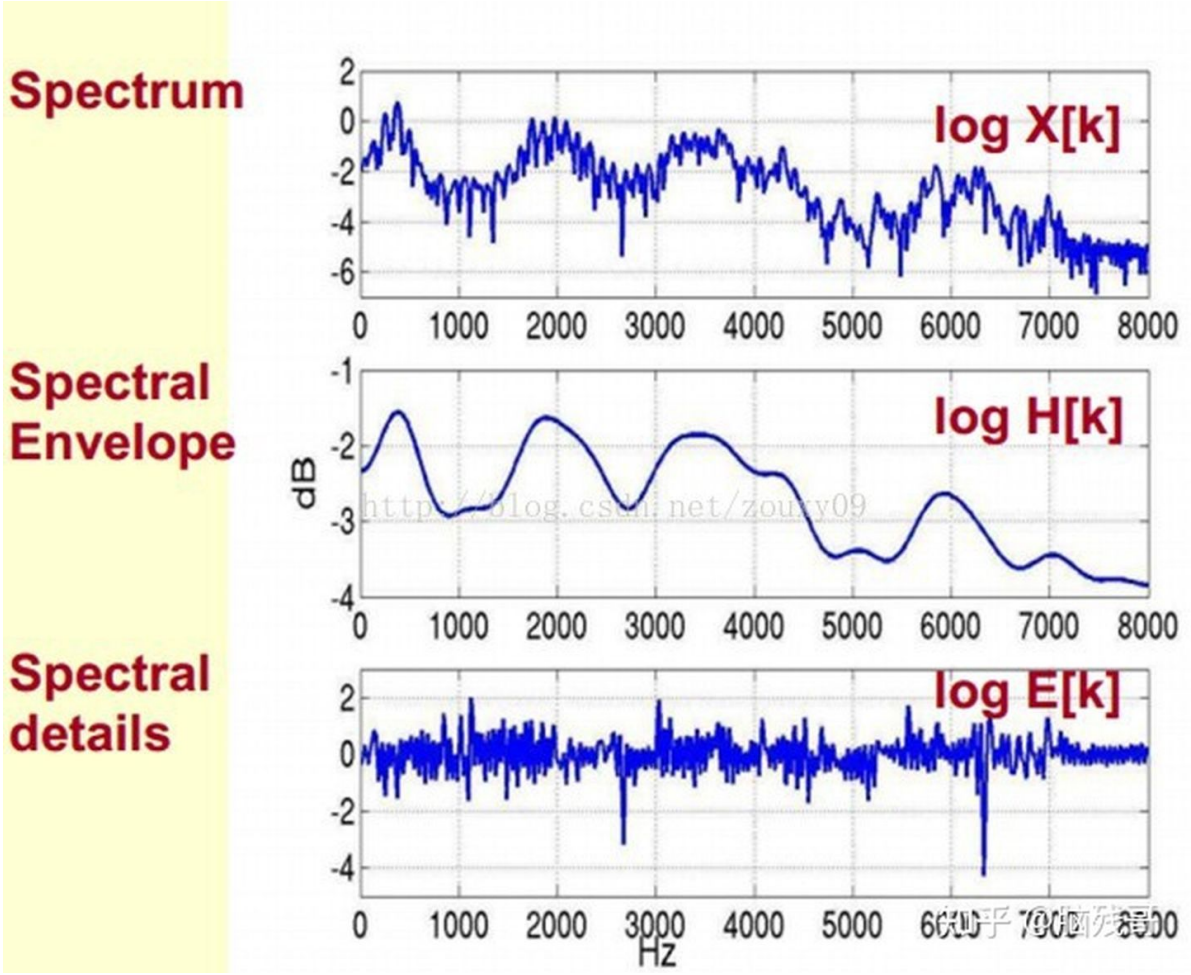
✓ 0.2s Python



可以看到，男生的频谱在低频域的分贝较大，女生的频谱在高频域的分贝较大，因此梅尔频谱是一个区分度较大的特征。

梅尔倒谱系数

如果我们对梅尔频谱看做是时域信号，先取对数，然后再做一次傅里叶变换（实际上是逆傅里叶变换）， 通过低通滤波器获得频谱包络，那么我们就获得了梅尔倒谱系数MFCC，MFCC就是这帧语音的特征。



所谓梅尔倒谱系数指的就是梅尔频谱的包络（Spectral Envelope）， 它可以概括梅尔频谱的变化趋势，作为音频的一大特征。

在实践中，我们使用librosa.feature.mfcc获得MFCC，其中参数n_mfcc表示MFCC的个数。

由于每个音频的时长不同，我们获得的梅尔倒谱系数个数也不同，这样会影响后续的模型架构。因此，为了统一特征的个数，我们在时间方向求平均值，获得n_mfcc个平均梅尔倒谱系数。

同样的，我们计算出男生和女生的梅尔倒谱系数，并且画出系数的直方图。

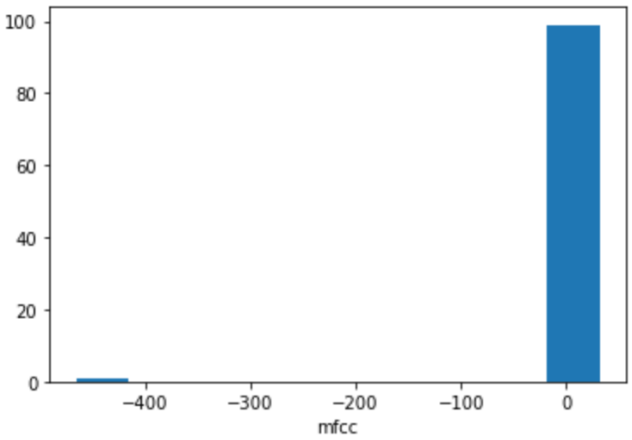
```python
# 女生的梅尔倒谱系数
y_ps, sr = librosa.load(f"data/train/1.wav", sr=None)
mfcc = np.mean(librosa.feature.mfcc(y=y_ps, sr=sr, n_mfcc=100, n_fft=1024, ho
print(f"max: {max(mfcc)}")
print(f"min: {min(mfcc)}")
```

✓ 0.7s Python

```
max: 32.2575569152832
min: -466.370361328125
```

```python
plt.hist(mfcc)
plt.xlabel("mfcc")
plt.show()
```

Python



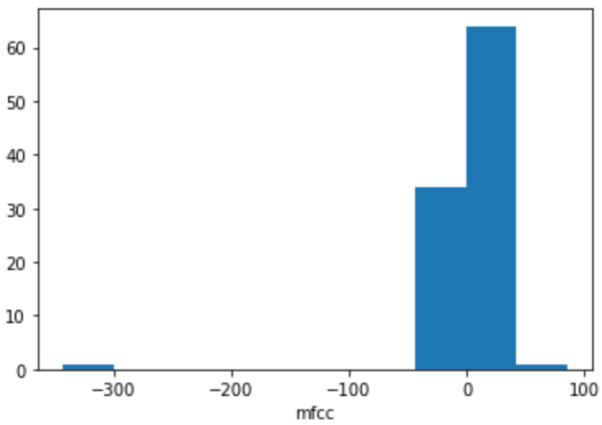

```python
# 男生的梅尔倒谱系数
y_ps, sr = librosa.load(f"data/train/4.wav", sr=None)
mfcc = np.mean(librosa.feature.mfcc(y=y_ps, sr=sr, n_mfcc=100, n_fft=1024, ho
print(f"max: {max(mfcc)}")
print(f"min: {min(mfcc)}")
```

[28] Python

```
max: 85.02069854736328
min: -343.28533935546875
```

```python
plt.hist(mfcc)
plt.xlabel("mfcc")
plt.show()
```

[25] Python



可以看到，男生和女生的梅尔倒谱系数分布存在一定的差异，可以作为音频特征。

特征工程

有了上述的数据探索，我们决定将平均基频和梅尔倒谱系数作为数据特征。

为了更好地刻画音频特征我们采用99个平均倒谱系数以及平均基频，一共100个特征作为模型的输入。

我们用pytorch的Dataset类定义数据集，每次从数据集中获取一项，我们都会调用librosa.feature.mfcc函数获取音频的MFCC以及平均基频作为特征。其中，平均基频已经在数据探索环节中计算出来并存储在excel文件中。代码详见dataset.py。

```
def __getitem__(self, index: int):
    data = self.data[index]
    audio_dir = os.path.join("data/train",data[0])
    label = data[1]
    bf = np.array([data[2]])
    mfcc = self.getmfcc(audio_dir)
    input = np.concatenate((mfcc, bf))
    return input, np.int64(label)
def getmfcc(self, wav_file_path):
    y_ps, sr = librosa.load(wav_file_path, sr=None, duration=2)
    mfcc = np.mean(librosa.feature.mfcc(y=y_ps, sr=sr, n_mfcc=99, n_fft=1024, hop_length=512), axis=1)
    return mfcc
```

模型架构

我们采用四层神经网络作为模型，这四层的架构为：

- 输入为100维，输出为200维，激活函数为ReLU
- 输入为200维，输出为200维，激活函数为ReLU
- 输入为200维，输出为200维，激活函数为ReLU
- 输入为200维，输出为2维，激活函数为Sigmoid

最后一层输出的2维结果是每一个类别的分类分数，最后我们会对其进行softmax操作，获得各个类别的概率。

代码详见model.py。

```
import torch
import torch.nn as nn

class NN(nn.Module):
    def __init__(self):
        super(NN, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(100, 200),
            nn.ReLU(),
            nn.Linear(200, 200),
            nn.ReLU(),
            nn.Linear(200, 200),
            nn.ReLU(),
            nn.Linear(200, 2),
            nn.Sigmoid()
        )
    def forward(self,input):
        return self.layers(input)
```

训练过程

训练、评价及测试代码详见main.ipynb，下面作简要描述。

首先，我们将数据集根据9:1的比例分为训练集和测试集（1800个样例作为训练集，200个样例作为测试集），切分过程采用random函数随机分组。

```python
index = [i for i in range(2000)]
random.shuffle(index)
train_index = index[:1800]
test_index =  index[1800:]
```

```python
train = AudioTrainDataset(config.train_data_dir, train_index)
test = AudioTrainDataset(config.train_data_dir, test_index)
```

为了加快训练速度，采用cuda平台进行训练。

```python
device = torch.device(config.device)
```

```python
model = NN().to(device)
```

损失函数采用交叉熵损失函数，优化器采用Adam优化器。

```python
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),lr = config.lr)
```

我们设置50个epoch进行训练，每次训练完1个epoch我们都会计算训练集的损失函数、正确率，并且会在测试集上测试一遍模型，计算测试集的准确率，精确率，召回率和F1得分。每当测试集的准确率有所上升，我们会保存一次模型，留到后续以填充真正的无标签测试集。

```
Epoch 43, train loss is 0.3335, train acc is 0.9792, test loss is 0.3535, test acc is 0.9600,
test precision is 0.9730, test recall is 0.9730, test f1 is 0.9730.
100%|██████████| 57/57 [00:08<00:00,  6.82it/s]
  2%|          | 1/57 [00:00<00:08,  6.97it/s]
Epoch 44, train loss is 0.3476, train acc is 0.9638, test loss is 0.3489, test acc is 0.9600,
test precision is 0.9861, test recall is 0.9595, test f1 is 0.9726.
100%|██████████| 57/57 [00:08<00:00,  6.45it/s]
  2%|          | 1/57 [00:00<00:10,  5.48it/s]
Epoch 45, train loss is 0.3292, train acc is 0.9846, test loss is 0.3565, test acc is 0.9550,
test precision is 0.9603, test recall is 0.9797, test f1 is 0.9699.
100%|██████████| 57/57 [00:09<00:00,  6.21it/s]
  2%|          | 1/57 [00:00<00:07,  7.03it/s]
Epoch 46, train loss is 0.3303, train acc is 0.9825, test loss is 0.3763, test acc is 0.9350,
test precision is 0.9299, test recall is 0.9865, test f1 is 0.9574.
100%|██████████| 57/57 [00:08<00:00,  6.67it/s]
  2%|          | 1/57 [00:00<00:08,  6.48it/s]
Epoch 47, train loss is 0.3295, train acc is 0.9836, test loss is 0.3630, test acc is 0.9500,
test precision is 0.9539, test recall is 0.9797, test f1 is 0.9667.
100%|██████████| 57/57 [00:08<00:00,  6.57it/s]
  2%|          | 1/57 [00:00<00:08,  6.33it/s]
Epoch 48, train loss is 0.3337, train acc is 0.9792, test loss is 0.3737, test acc is 0.9350,
test precision is 0.9245, test recall is 0.9932, test f1 is 0.9577.
100%|██████████| 57/57 [00:08<00:00,  6.56it/s]
  2%|          | 1/57 [00:00<00:08,  6.63it/s]
Epoch 49, train loss is 0.3342, train acc is 0.9786, test loss is 0.3522, test acc is 0.9600,
test precision is 0.9730, test recall is 0.9730, test f1 is 0.9730.
100%|██████████| 57/57 [00:08<00:00,  6.53it/s]
Epoch 50, train loss is 0.3303, train acc is 0.9830, test loss is 0.3551, test acc is 0.9600,
test precision is 0.9545, test recall is 0.9932, test f1 is 0.9735.
```

训练的具体参数详见config.py。

评价指标

我们采用4个评价指标。

准确率（accuracy）

指的是预测标签与真实标签相同的比例。

精确率（precision）

$$Precision \triangleq \frac{TP}{TP+FP}$$

从预测结果角度出发，描述了二分类器预测出来的正例结果中有多少是真实正例，即该二分类器预测的正例有多少是准确的。

召回率（recall）

$$Recall \triangleq \frac{TP}{TP+FN}$$

从真实结果角度出发，描述了测试集中的真实正例有多少被二分类器挑选了出来，即真实的正例有多少被该二分类器召回。

F1得分（F1 score）

$$F1 = \frac{2(precision \times recall)}{(precision + recall)}$$

模型性能

经过上述训练过程，我们获得了最终的模型。我们对最终模型在测试集上进行了上述4个评价指标的评估。

```
test loss is 0.3454, test acc is 0.9700, test precision is 0.9863, test recall is 0.9730, test f1 is 0.9796.
test confusion matrix:
[[ 50   4]
 [  2 144]]
```

模型的准确率为97%，精确率为98.63%，召回率为97.3%，F1得分为97.96%。从混淆矩阵可以看到在200个样例中仅有6个样例分类错误。

填充测试集

最后，我们用训练好的模型对测试集进行标签的填充，填充结果详见test.xlsx。

```python
model.eval()
total_y = []
df = pd.read_excel(config.test_data_dir)
filename = list(df['filename'])
bf = list(df['bf'])
for data in testloader:
    data = data.to(torch.float32).to(device)
    output = model(data)
    y_ = output.data.cpu().numpy()
    y_ = np.argmax(y_,axis = 1)
    total_y.extend(list(y_))
l = []
for i in range(df.shape[0]):
    l.append([filename[i],total_y[i], bf[i]])
```

Python

```python
res = pd.DataFrame(l, columns=['filename', 'label', 'bf'])
res.to_excel(config.test_data_dir, index=None)
```

Python

filename	label
2001.wav	0
2002.wav	1
2003.wav	1
2004.wav	1
2005.wav	1
2006.wav	1
2007.wav	1
2008.wav	1
2009.wav	1
2010.wav	1
2011.wav	0
2012.wav	1
2013.wav	1
2014.wav	0
2015.wav	0
2016.wav	1
2017.wav	0
2018.wav	1
2019.wav	1
2020.wav	1