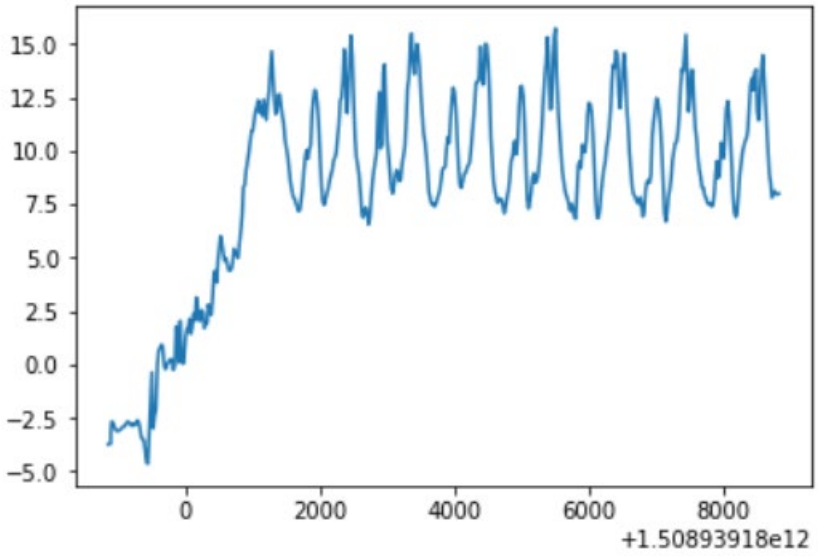
基于加速度传感器的步态识别行人分类实验

学生姓名	董开宇	学号	16211130	指导老师	李辉勇
实验地点	F332	实验时间	4 月 16 日	班级	162115

一、 数据处理

首先以时间为横轴，以 x 方向加速度大小为纵轴绘制曲线，对数据有个大概的预览：

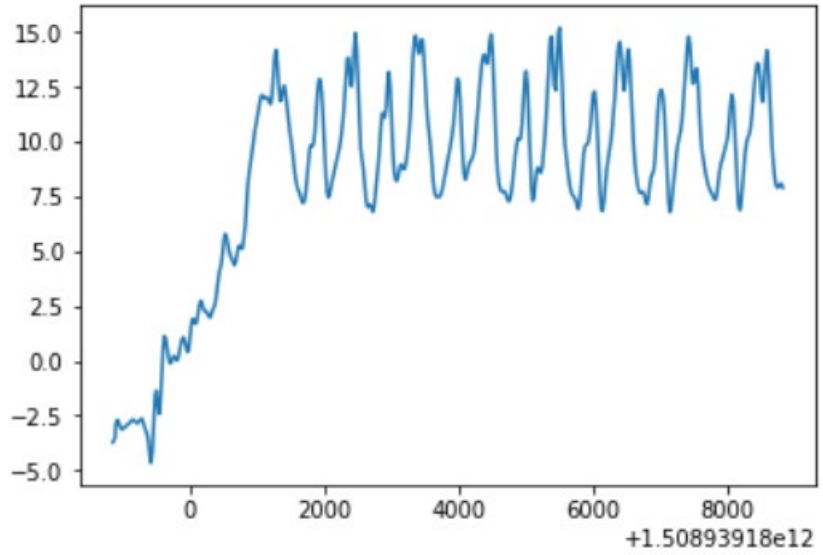


1、滤波：

```
#滤波
b,a = signal.butter(4,0.4,'lowpass')
ax = signal.filtfilt(b, a, ax)
ay = signal.filtfilt(b, a, ay)
az = signal.filtfilt(b, a, az)
```

由实验说明得知，对原始数据进行滤波可以有效地去除噪声平滑曲线，但并没有详细介绍关于滤波器如何应用在本数据集。因为不知道对于步态加速度信号具体滤波参数如何设置，于是参考了相关论文[1]得知：人在行走过程中加速度信号的大部分频谱都低于 4Hz，因此使用具有 4 Hz 截止频率的 4 阶低通巴特沃斯滤波器进行滤波。

滤波结果：



可以看到通过 4 Hz 截止频率的 4 阶低通巴特沃斯滤波器后，数据噪声明显减少，但并未丢失图线中的波峰波谷信息。

2、插值：

由实验说明得知，每条数据之间的间隔并不是严格的 20ms，为了消去时间这一特征值，下面进行插值，将数据间隔设置为 20ms。

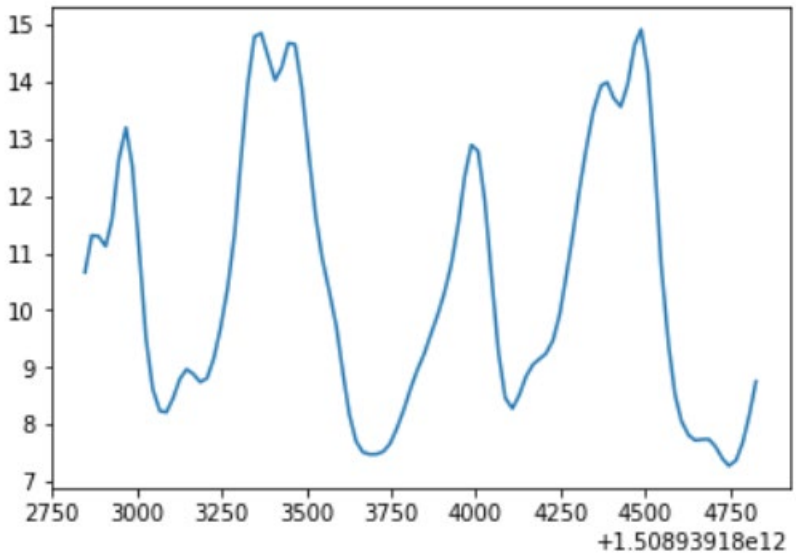
插值函数参数选择为 cubic，即三次样条插值：

```
#三次样条插值
fx=interp1d(time,ax,kind='cubic')
fy=interp1d(time,ay,kind='cubic')
fz=interp1d(time,az,kind='cubic')
```

每隔 20ms 进行插值：

```
for j in range(1,28500):
    new_time.append(new_time[j-1]+20)
    new_ax.append(fx(new_time[j-1]+20))
    new_ay.append(fy(new_time[j-1]+20))
    new_az.append(fz(new_time[j-1]+20))
    preprocess.append([new_ax[j],new_ay[j],new_az[j]])
```

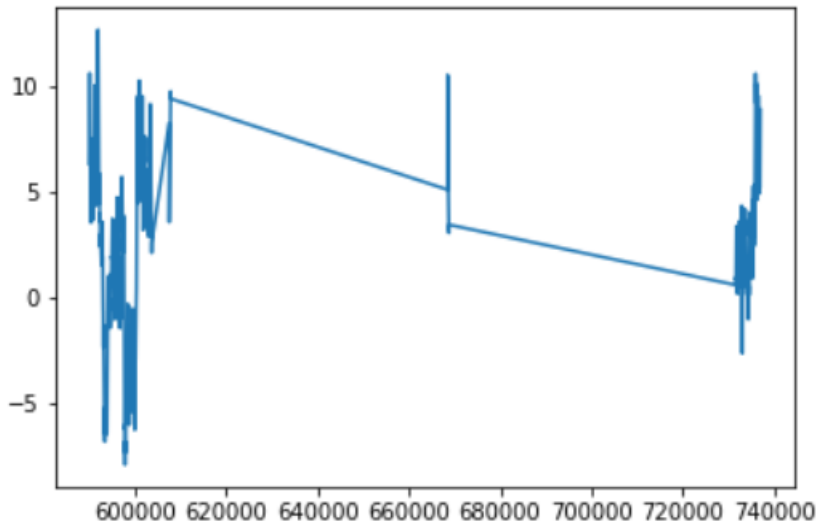
滤波插值后的结果如图：



3、丢弃异常值数据：

通过预览数据时查看曲线关系，发现在刚开始计时加速度数据有异常波动。于是同理查看一下结束阶段是否也有异常波动：

```
1  plt.plot(time[-1000:],ax[-1000:])
2  plt.show()
```



显然这些数据是人未处于稳定的行走阶段的数据，不能由于训练和预测识别人的身份，故丢弃这些异常值。

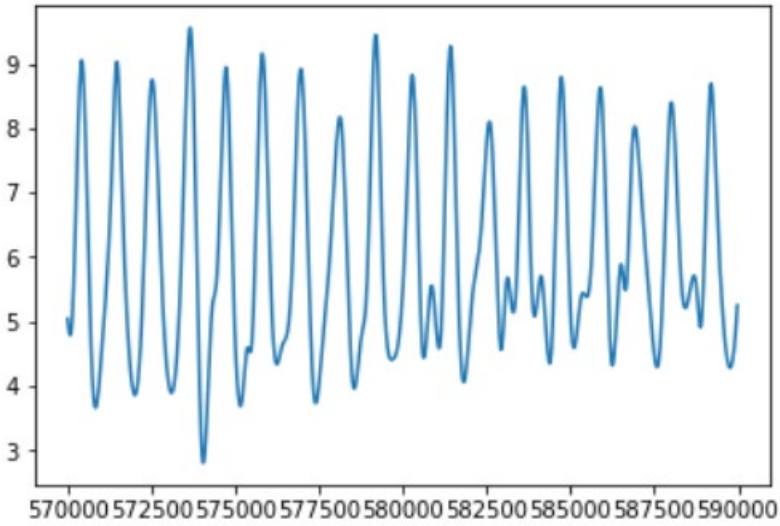
说明：总数据条数为 30500 条，从第 1000 条数据开始取用连续 28500 条，结果上看即为丢弃了前 1000 条与后 1000 条可能有异常波动的数据。

```
#丢弃前1000条包含异常值的数据
new_time = [time[1000]]
new_ax = [fx(new_time)]
new_ay = [fy(new_time)]
new_az = [fz(new_time)]

preprocess=[]
preprocess.append([new_ax[0][0],new_ay[0][0],new_az[0][0]])

for j in range(1,28500):
    new_time.append(new_time[j-1]+20)
    new_ax.append(fx(new_time[j-1]+20))
    new_ay.append(fy(new_time[j-1]+20))
    new_az.append(fz(new_time[j-1]+20))
    preprocess.append([new_ax[j],new_ay[j],new_az[j]])
```

过程中曾经尝试了 UnivariateSpline 方法实现去噪插值的同时进行，但由于对参数设置不够熟悉，最终得到的曲线过于平滑，丢失了一些局部的波峰和波谷信息，效果不太理想，遂放弃。



4、数据分组

使用了比较简单的无重叠滑动窗口分组方法，窗口大小为 LINES_CHUNK

```
#分组
def chunkify(fname, lines=LINES_CHUNK):
    with open(fname, 'r') as f:
        #跳过第一行index
        next(f)

        for chunk in zip(*[f] * lines):
            yield chunk
```

5、特征提取

所给数据的特征仅有三个坐标轴的加速度大小和时间，通过插值处理将时间这一特征消去，之后通过一些方法由三轴加速度大小计算得到新的特征：

平均值、标准差、绝对离差、矢量合成、过零率、最小值、最大值

```
#平均值
means = ft._avg(sensorVals)
results.extend([means[0], means[1], means[2]])

#标准差
results.extend(ft._stddev(sensorVals, means))

#绝对离差
results.extend(ft._absdev(sensorVals, means))

#矢量合成
results.append(ft._resultant(sensorVals))

#过零率
results.extend(ft._zerocross(sensorVals))

#最小值
results.extend(ft._minima(sensorVals))

#最大值
results.extend(ft._maxima(sensorVals))

#类别标签
results.append(i)
```

6、归一化

```
#归一化
df[NORM_COLS] = df[NORM_COLS].apply(lambda x: (x - x.min()) / (x.max() - x.min()))

df.to_csv('proc.csv', encoding='utf-8', index=False, quoting=csv.QUOTE_NONNUMERIC)
```

二、 实验过程

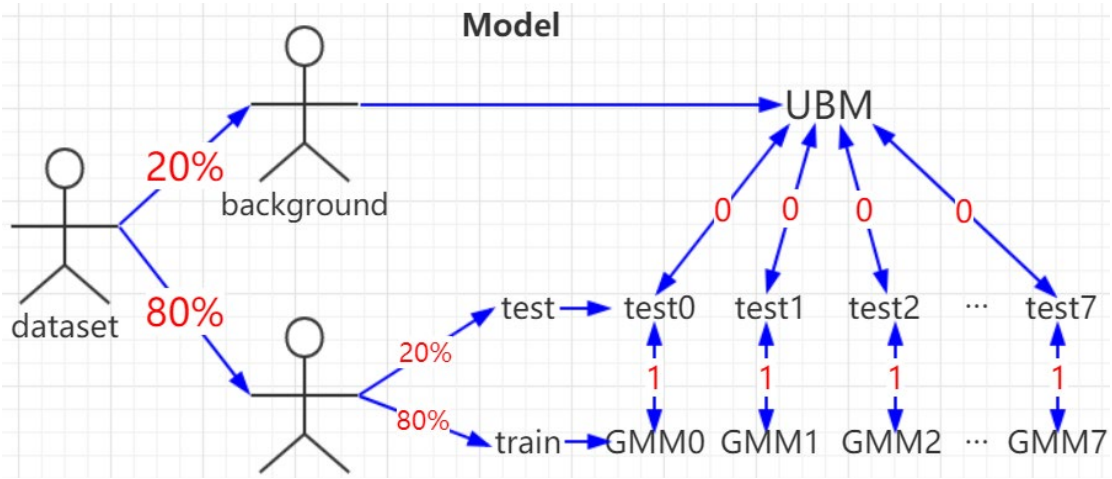
1、算法选择

之前和同学组队参加未来杯竞赛语音赛道时的作品使用了基于 GMM-UBM 的说话人识别算法[2]，声音的波形数据与这次步态数据看上去比较类似，所以本次实验我也选用了 GMM-UBM（高斯混合模型-通用背景模型）算法来实现对行人的分类识别，从结果上看，将 GMM-UBM 模型应用于步态识别是完全可行的。

GMM 将空间分布的概率密度用多个高斯概率密度函数的加权来拟合，可以平滑的逼近任意形状的概率密度函数，并且是一个易于处理的参数模型，具备对实际数据极强的表征力。但反过来，GMM 规模越庞大，表征力越强，其负面效应也会越明显：参数规模也会等比例的膨胀，需要更多的数据来驱动 GMM 的参数训练才能得到一个更加通用（或称泛化）的 GMM 模型。

GMM-UBM 实际上是一种对 GMM 的改进方法。将选取的背景数据混合起来充分训练出一个 GMM，这个 GMM 可以看作是对行人步态数据的表征，但是又由于它是从其他不止一人的混杂数据中训练而成，它又不具备表征具体身份的能力。

2、模型结构



3、实验思路

首先从数据集中分出 20%用于训练背景模型：

```
users = [i for i in range(10)]
background = users[-BG_SIZE:]
X_background = df.loc[df['class'].isin(background)]
X_background = X_background.drop(X_background[['class']], axis=1)

#训练背景模型
gmm_bg = GaussianMixture()
gmm_bg.fit(X_background)
```

对于剩下 80%的数据，按 80%：20%的比例划分训练集和测试集。

```
for u in users[:-BG_SIZE]:
    user_data = df[df['class'] == u]
    x = user_data.drop(user_data[['class']], axis=1)

    X_train, X_test = train_test_split(x, random_state=SEED)
```

训练集用于训练每个人的步态行人识别模型。

```
#训练步态行人识别模型
gmm = GaussianMixture()
gmm.fit(X_train)
```

测试集用来在之前训练好的背景模型和刚才训练好的步态行人识别模型上分别计算加权对数概率作为评分。

```
# 计算加权对数概率作为评分
pos_scores = pd.DataFrame(columns=['score', 'label', 'class'])
pos_scores['score'] = gmm.score_samples(X_test)
pos_scores['label'] = '1'
pos_scores['class'] = u
scores = scores.append(pos_scores)

neg_scores = pd.DataFrame(columns=['score', 'label', 'class'])
neg_scores['score'] = gmm_bg.score_samples(X_test)
neg_scores['label'] = '0'
neg_scores['class'] = u
scores = scores.append(neg_scores)
```

最后是结果的可视化：

```
plot_AUC(scores)
```

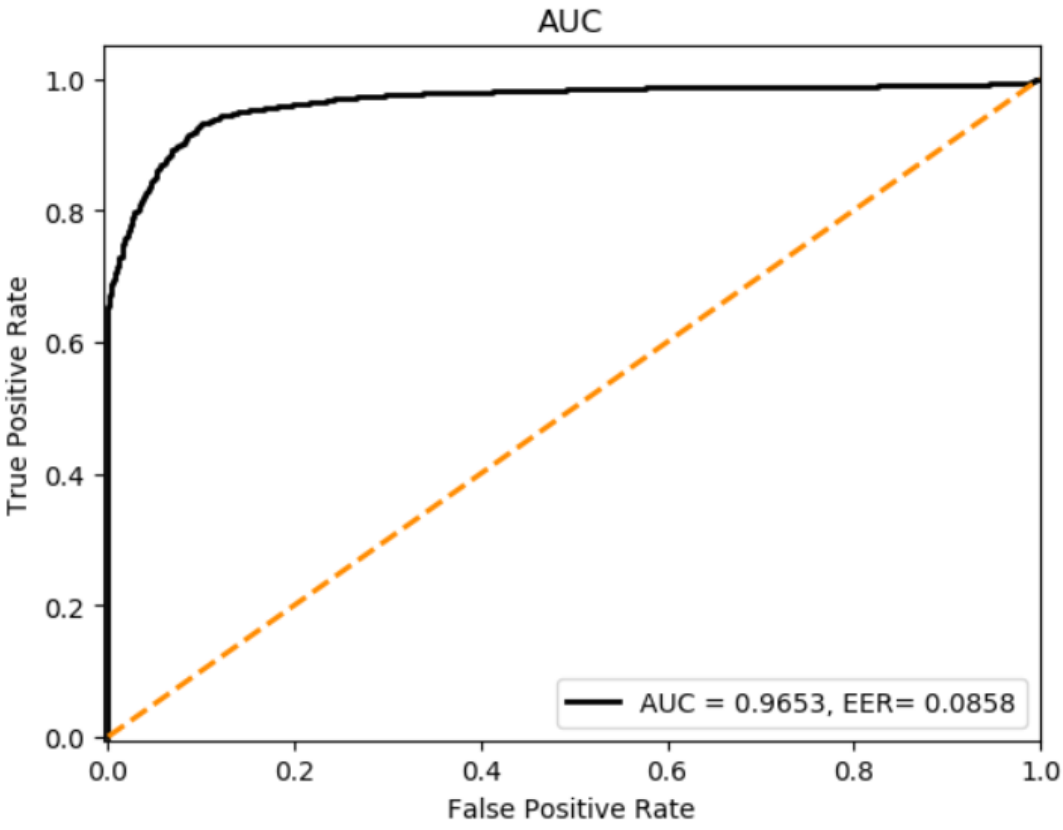
```
def plot_AUC(df):
    labels = df['label']
    scores = df['score']
    labels = [int(e)   for e in labels]
    scores = [float(e) for e in scores]
    auc_value = roc_auc_score(labels, scores)
    fpr, tpr, thresholds = roc_curve(labels, scores, pos_label=1)
    eer = brentq(lambda x: 1. - x - interp1d(fpr, tpr)(x), 0., 1.)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='black', lw=lw, label='AUC = %0.4f, EER= %0.4f' % (auc_value, eer))
    plt.plot([0, 1], [0, 1], color='darkorange', lw=lw, linestyle='--')
    plt.xlim([-0.005, 1.0])
    plt.ylim([-0.005, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('AUC')
    plt.legend(loc="lower right")
    plt.show()
    return
```

4、参数的调整

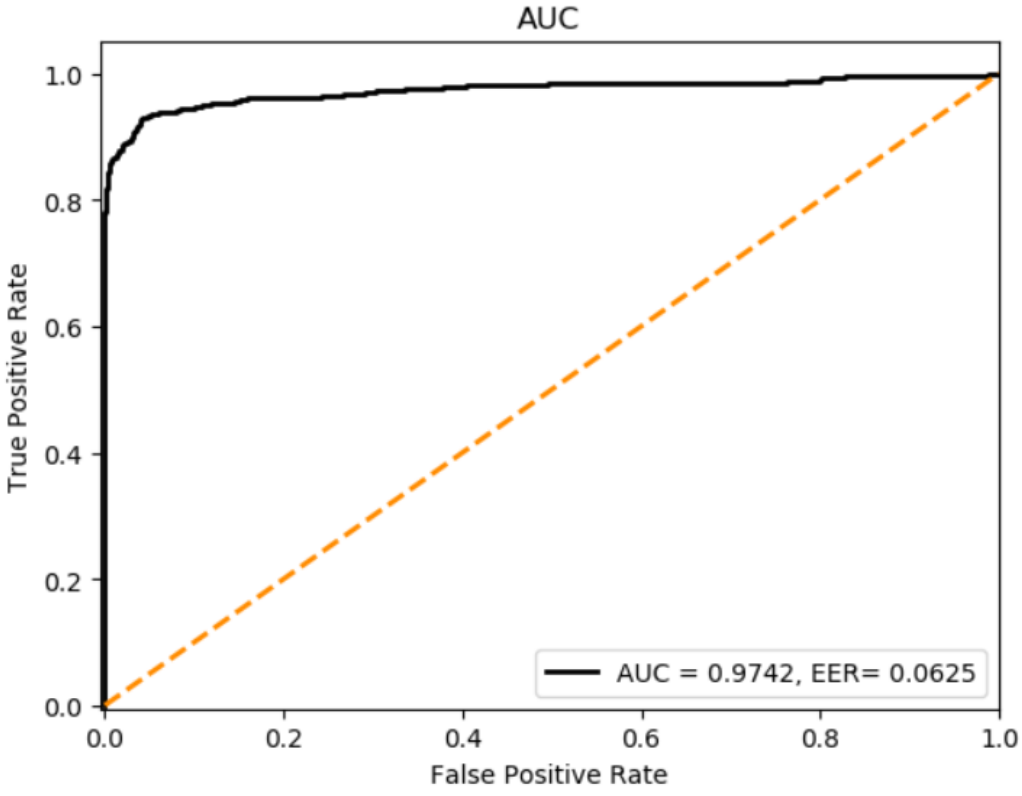
使用曲线下面积（AUC）和等误差率（EER）测量模型的性能。

发现窗口大小（CHUNK_LINE）对结果影响较为明显：

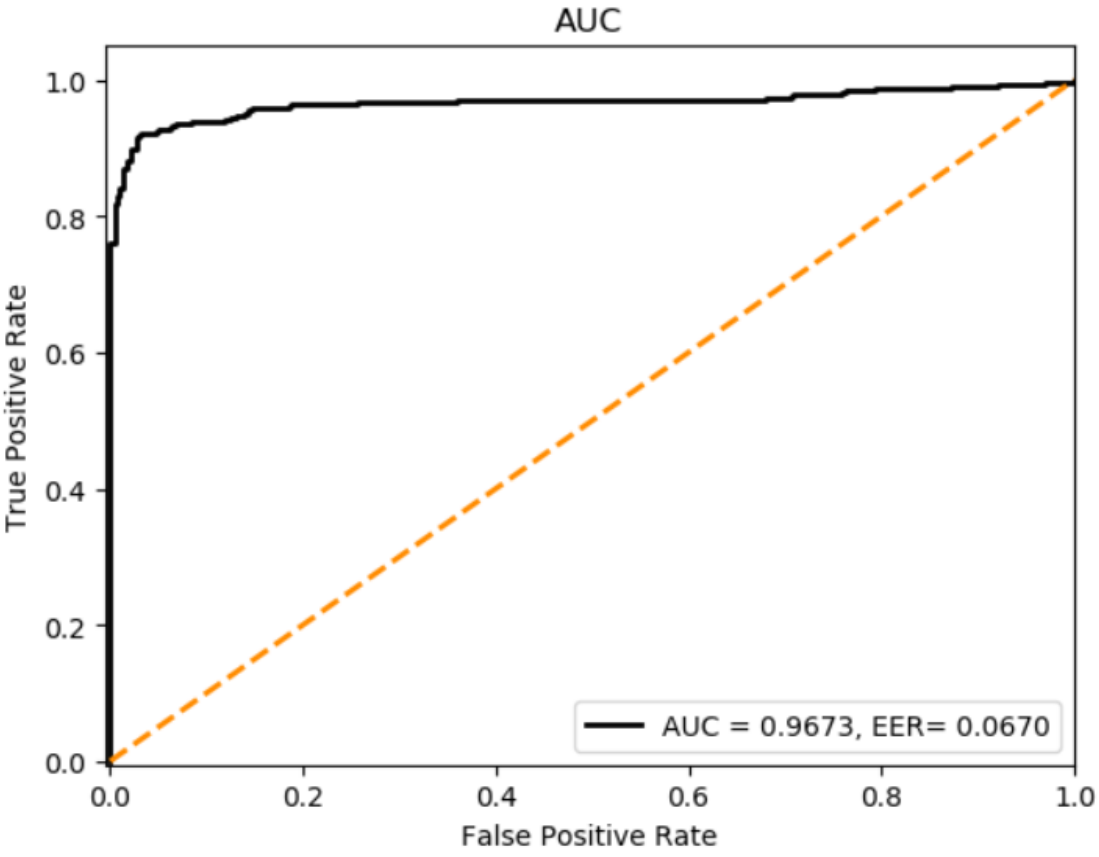
CHUNK_LINE=32:



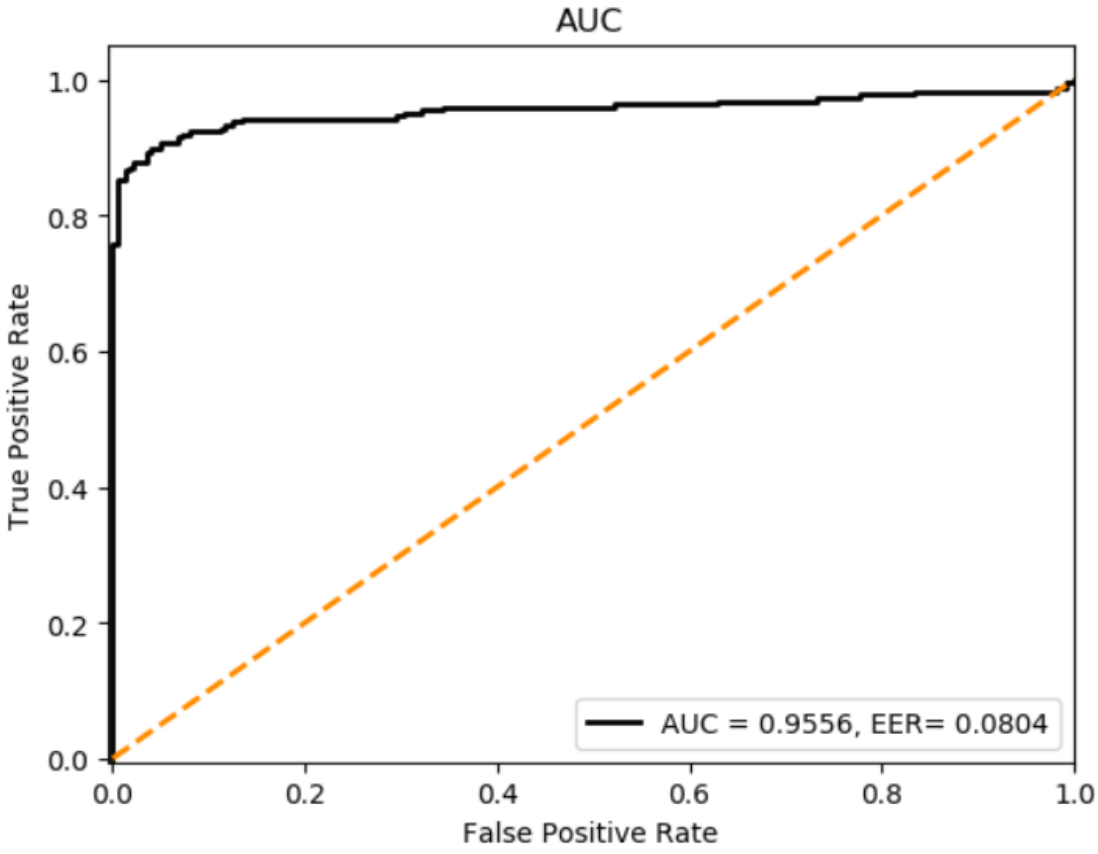
CHUNK_LINE=64:



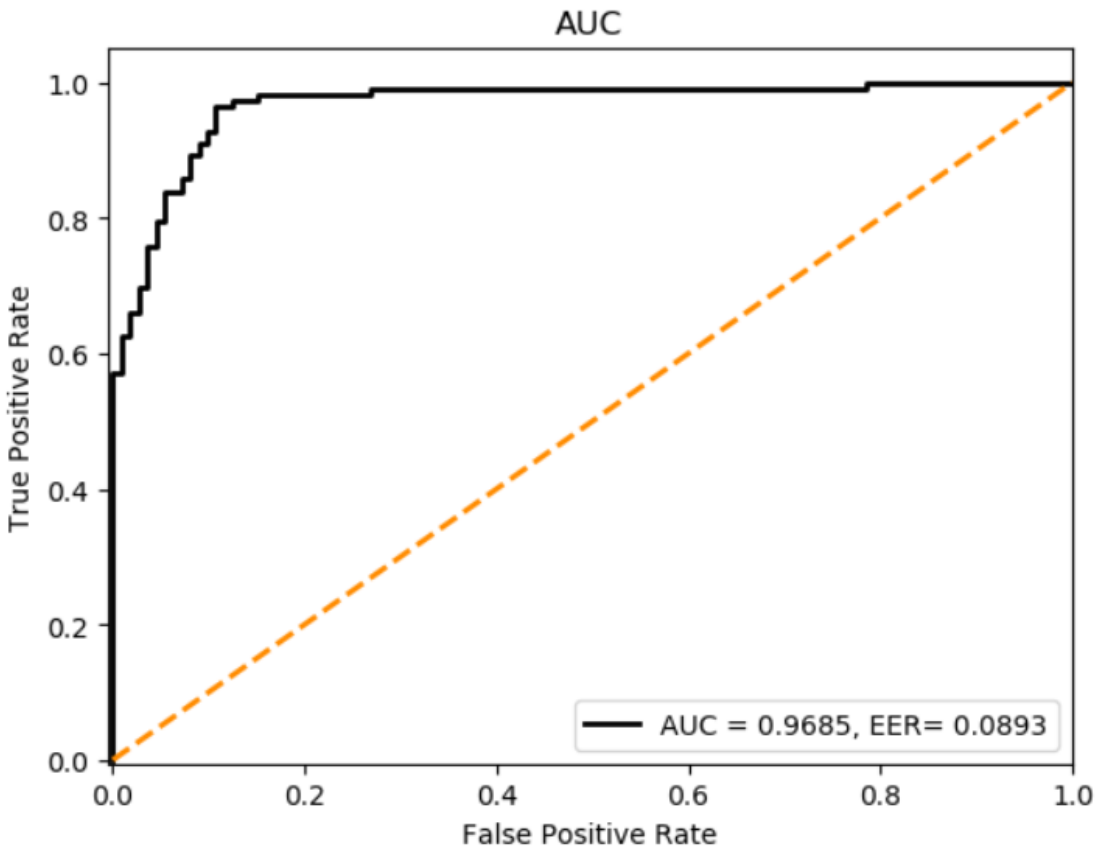
CHUNK_LINE=128:



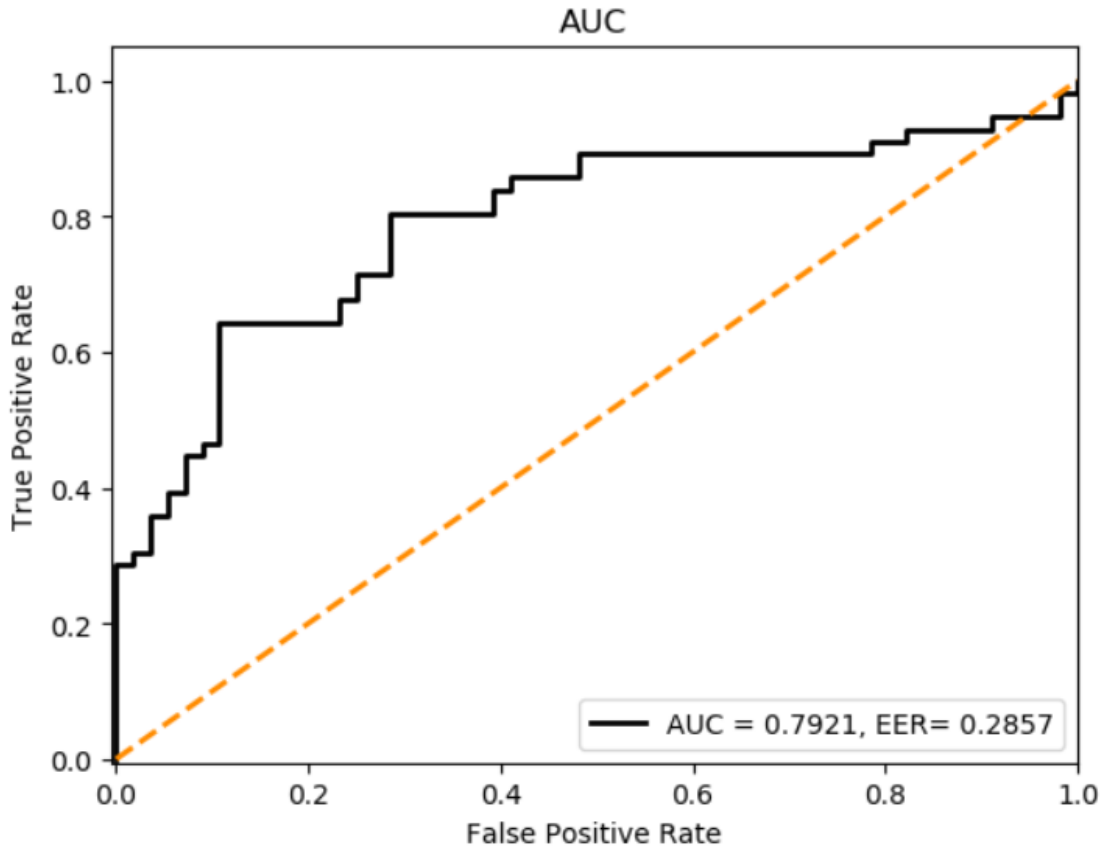
CHUNK_LINE=256:



CHUNK_LINE=512:

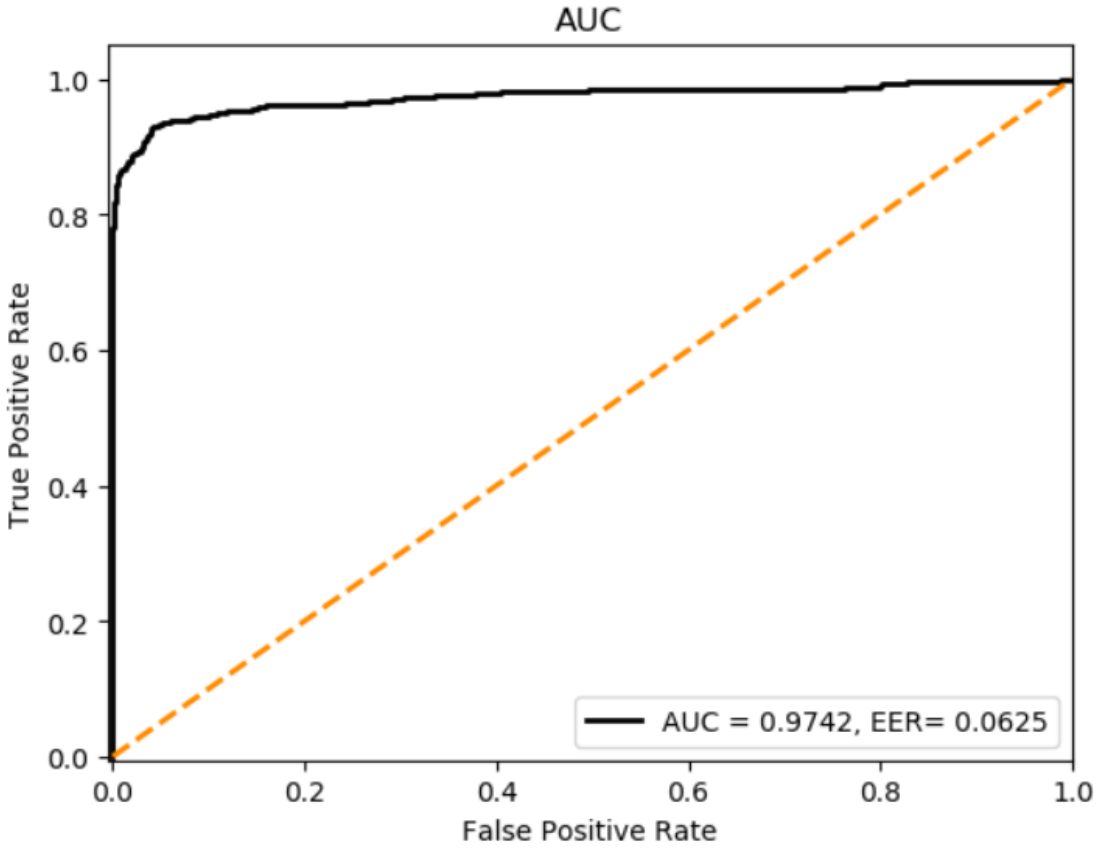


CHUNK_LINE=1024:



三、 结果及结论

在这个数据集上，窗口大小为64时的结果最优：
曲线下面积AUC=97.42%，等错误率EER=93.75%



将常用于说话人语音识别的GMM-UBM算法应用于步态行人识别分类是完全可行的，且模型效果较为良好。

四、 程序说明

目录及文件结构：

./data 文件夹内保存的是原始数据集，同时初步预处理的中间文件也生成在这个目录下。

./pic 文件夹里是我在实验过程中保存的通过调整窗口大小 CHUNK_LINE 这个参数得到的不同结果的图片。

./util 文件夹中存放主函数用到的文件，其中 const.py 内定义了所有用到的常量，features.py 中是进行特征提取的函数，process.py 里是进行数据处理的函数，visualize.py 中定义了用于结果可视化的函数。

根目录下 main.py 文件为程序入口，proc.csv 为数据被处理完毕后变成可被算法接收的形式同时生成的处理过后的数据文件。

五、 参考文献

[1] Tong K , Granat M H . A practical gait analysis system using gyroscopes[J]. Medical Engineering and Physics, 1999, 21(2):87-94.

[2] 基于 GMM-UBM 的说话人识别算法