# 睡眠分期大作业

### 任务描述

依据AASM 标准，使用课程中给到的ECG数据提取特征并神经网络模型实现睡眠分期。

### 模型选择说明

这里选择搭建**一维CNN神经网络模型**来完成这个任务。原因有以下几点：

1. 根据ECG进行睡眠分期是一个有监督的多分类任务，而神经网络模型很适合完成这个任务。
2. 由阅读的论文资料得知，目前CNN是基于ECG的睡眠任务中最常用和最安全的DL模型，虽然这种“常用”是因为CNN很适合学习ECG数据中的局部特征（P波Q波等）并很好的建模，但同时CNN也能从手动提取的统计特征中捕捉到的抽象规律，然后利用这些规律进行睡眠阶段的分类，本次作业中选择CNN神经网络也是因为后者的关系。
3. CNN神经网络在计算上更高效。

同时这里并没有采用原始ECG数据输入神经网络中进行特征提取，而是进行手动特征提取再输入给CNN神经网络模型，这是因为：

1. 原始数据量只有1084个Epoch（也就是1084 \* 30 \* 125）我担心数据量比较少。同时原始数据具有很高的信息密度，包含丰富的时域和频域信息。为了让模型更快地收敛，这里选择了进行手动提取特征。

### 拟用算法步骤

1. 读取ECG.txt和label\_profusion.txt获取原始ECG数据和对应的标签
2. 对ECG数据手动提取其统计特征
3. 按照8：2的比例划分训练集和测试集，并定义loss函数和优化器
4. 将提取到的训练统计特征输入一维CNN网络进行学习
5. 在测试集上用深度网络进行推断，通过一些评价指标对模型的能力进行定量评估
6. 评估结果的输出与可视化

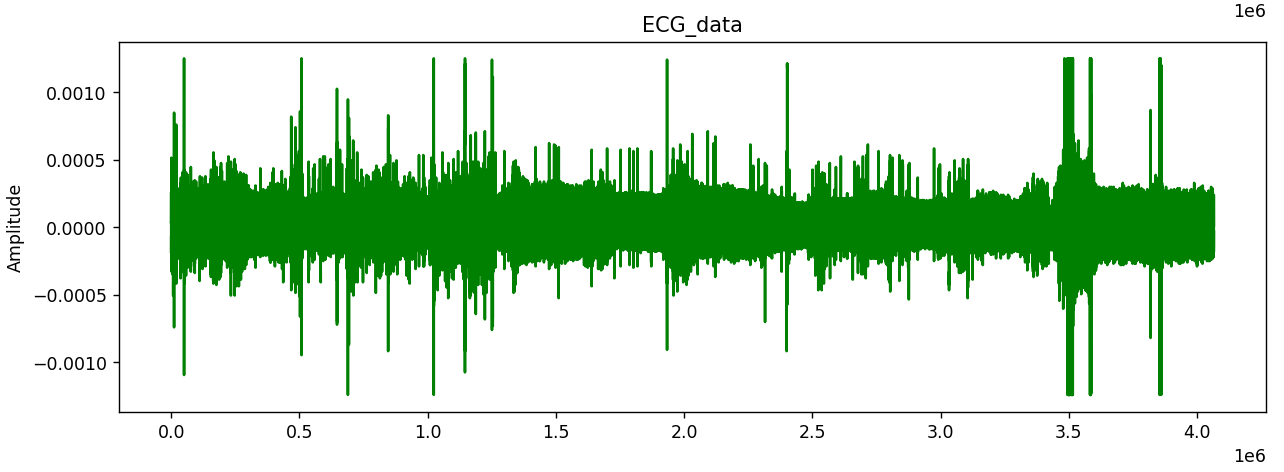
### 数据描述

ECG.txt文件是记录的心电图数据，其包含了4065000个浮点数，采样频率是125hz，大约采集了9h的心电图数据。每个浮点数代表一个时间点上的心脏电压值。数据每30s划分为一个epoch，一共 1084个epoch，用于进行睡眠分期分类。

label\_profusion.txt文件是ECG.txt数据对应的睡眠分期分类的文件，其包含1084个整形数（只有0，1，2，3，4这个五个数），每个数顺序对应ECG.txt中一个epoch的睡眠分期标签，睡眠分期标签的映射如下：w：0 ；N1：1； N2：2；N3：3；R：4

### 原始数据可视化

纵轴表示某时间点的心脏电压值，横轴是采样次数（125hz采样频率），一共约9个小时



### 预处理说明

实际上**没有对原始ECG数据进行任何预处理操作**，原因如下：

**缺省**：

首先由于数据没有缺省，因此不需要进行对应的缺省预处理。

**去噪：**

去噪使用了巴特沃斯带通滤波器对过低或者过高的频率范围进行了抑制操作，然而经过不断调整低频高频的阈值发现，不管怎么设置阈值，去噪后的ECG数据都会导致CNN神经网络学习能力下降（固定了种子，观察到最终指标变差），**因此最终没有采用去噪处理**（仍然可以在Utils文件中找到函数）。

**归一化处理**：

使用了简单的min-max归一化将ECG数据归一到-1,1的区间（注意这里不能归一到0,1的区间，会丢失大量信息），也使用了z\_score标准化将ECG数据归一到-1,1的区间。然而经过测试发现归一化后CNN神经网络学习能力没有改变（固定了种子，观察到最终指标基本没有发生变化，z\_score会变差，因为破坏了特征信息），**因此最终没有采用归一化处理**（仍然可以在Utils文件中找到函数）。

同时也**没有对提取得到的特征进行特征的预处理操作**，原因如下：

**归一化处理**：

对提取完的特征使用了min-max归一化处理和z\_score归一化处理，然而经过测试发现归一化后CNN神经网络学习能力变差了（min-max在acc降低了大约8%，z\_score降低了大约11%），**因此最终没有采用归一化处理**（仍然可以在Utils文件中找到函数）。

### 特征提取

这里手动提取完特征后得到的特征变量features的大小是(1084,31)，也就是说特征维度的大小是31。由于处理特征是对每个epoch的数据进行处理得到特征。因此假设某个epoch对应的原始数据为{}，这里为了方便设，同时为了方便起见用来代表计算得到的特征，其中i是特征序号（譬如代表某个epoch计算得到的信号均值），小波变换使用的计算方法是

coeffs = pywt.wavedec(raw\_signal, 'db1', level=5)

然后根据得到的小波参数计算其统计特征，x对应尺度的小波系数为，该尺度小波系数总数为。下表给出了手动提取的特征的具体内容和计算公式：

|  |  |  |
| --- | --- | --- |
| 特征序号 | 物理含义 | 计算公式 |
| 0 | 信号均值 |  |
| 1 | 信号标准差，代表信号的离散程度 |  |
| 2 | 信号最小值 |  |
| 3 | 信号最大值 |  |
| 4 | 信号斜度，表示信号整体趋势 |  |
| 5 | 信号能量 |  |
| 6 | 功率谱密度总和 | 对进行傅里叶变换得到  进而得到 |
| 7+4x | 小波系数:均值 |  |
| 8+4x | 小波系数:标准差，代表小波系数的离散程度 |  |
| 9+4x | 小波系数:偏度，代表分布的偏斜程度 |  |
| 10+4x | 小波系数:峰度，代表分布的尖峭或平坦程度 |  |

需要注意的是，表格里特征序号的x是[0,5]之间的整数，最后的四种小波系数是对信号做了不同尺度的小波变换下得到的小波系数统计特征。

### 计算原理（模型结构）

# CNN模型

class SleepStageModel(nn.Module):

    def \_\_init\_\_(self, input\_size, num\_classes):

        super(SleepStageModel, self).\_\_init\_\_()

        self.cnn1d = nn.Sequential(

            nn.Conv1d(in\_channels=1, out\_channels=64, kernel\_size=3, stride=1, padding=1),

            nn.ReLU(),

            nn.MaxPool1d(kernel\_size=2, stride=2),

            nn.Conv1d(in\_channels=64, out\_channels=128, kernel\_size=3, stride=1, padding=1),

            nn.ReLU(),

            nn.MaxPool1d(kernel\_size=2, stride=2)

        )

        self.fc = nn.Linear(128 \* (input\_size // 4), num\_classes)

    def forward(self, x):

        x = x.float()  # 将输入数据类型转换为 torch.float32, x.shape = (32,1,31)

        x = self.cnn1d(x)

        x = x.view(x.size(0), -1)  # 展平

        x = self.fc(x)

        return x

模型将输入的提取到的特征进行卷积+激活+最大池化提取全局特征，最后输入全连接层直接分类。Loss采用交叉熵，优化器使用Adam，lr=0.001，训练测试比为8:2。计算步骤具体如下：

1. 输入已经提取好的特征x, x.shape = (bs,1,31)，其中bs是batch\_size不管，1是预留的通道数，31就是我们提取的特征。首先将float64的特征转换为float32的特征不然会出错。
2. 特征x进入self.cnn1d()，首先是一维卷积，参数如上所示，卷积后输出(bs,64,31)
3. 激活+池化，输出(bs,64,16)，这是神经网络一般管用的手法，在卷积后加激活和池化，卷积处理数据的过程中channel数量变多，原始维度变少。
4. 同样的结构，这次又是卷积，卷积后输出(bs,128,16)
5. 激活+池化，输出(bs,128,8)
6. 在卷积提取完全局特征后将输出展开得到(bs,128\*8)
7. 输入全连接层，输出(bs,5)
8. 最后对5那一维度取argmax得到预测得到的睡眠分期序号(0~4)

这里的计算公式可以参考神经网络基础的计算公式，不再过多赘述。

### 参数调优

为了让模型充分训练，考虑最终选择epoch为80.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Epoch** | accuracy | precision | recall | F1 score | ROC\_auc |
| 10 | 0.820 | 0.919 | 0.820 | **0.883** | 0.908 |
| 20 | 0.834 | **0.970** | 0.834 | 0.840 | 0.920 |
| 40 | **0.876** | 0.958 | **0.876** | 0.858 | 0.901 |
| 80 | **0.876** | 0.950 | **0.876** | 0.872 | 0.885 |
| 160 | 0.857 | 0.916 | 0.857 | **0.902** | **0.932** |

在搭建CNN神经网络模型的时候，直接采用了(conv1d + relu + maxpool) \* N的结构，这里的N尝试了1，2，3，4最终发现在重复两次（N=2）时效果较好，于是选用了重复两遍的结构。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | accuracy | precision | recall | F1 score | ROC\_auc |
| 1 | 0.853 | 0.920 | 0.853 | **0.901** | 0.927 |
| 2 | **0.876** | **0.950** | **0.876** | 0.872 | 0.885 |
| 3 | 0.866 | 0.949 | 0.866 | 0.875 | **0.928** |
| 4 | 0.825 | 0.856 | 0.825 | 0.862 | 0.710 |

同时在kernel\_size也进行了调参，结果发现size=3效果更好，并且理论上size=3的话参数利用率最高。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **kernel\_size** | accuracy | precision | recall | F1 score | ROC\_auc |
| 3 | **0.876** | **0.950** | **0.876** | 0.872 | 0.885 |
| 5 | 0.866 | 0.945 | 0.866 | **0.874** | **0.914** |

Channel的个数也进行了调参，发现channel: 1->64->128相对效果最佳，并且channel一直增大的话效果反而会变差。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Channel** | accuracy | precision | recall | F1 score | ROC\_auc |
| 1->32->64 | 0.857 | 0.920 | 0.857 | **0.904** | **0.936** |
| 1->64->128 | **0.876** | **0.950** | **0.876** | 0.872 | 0.885 |
| 1->128->256 | 0.853 | 0.936 | 0.853 | 0.857 | 0.922 |
| 1->256->512 | 0.848 | 0.919 | 0.848 | 0.900 | 0.927 |

### 调试分析

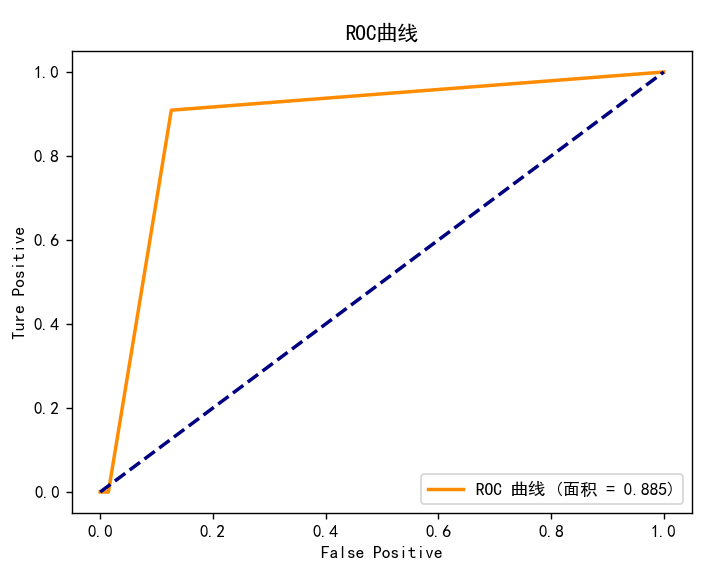
|  |
| --- |
| Traceback (most recent call last):  File "main.py", line 21, in <module>  ECG\_data = torch.tensor(ECG\_data)  ValueError: At least one stride in the given numpy array is negative, and tensors with negative strides are not currently supported. (You can probably work around this by making a copy of your array with array.copy().) |
| 某个变量使用了索引，比如x=x[...,::-1]，需要加.copy()来保证正常运行  ECG\_data = torch.tensor(ECG\_data)改为  ECG\_data = torch.tensor(ECG\_data).copy()后即可正常运行 |

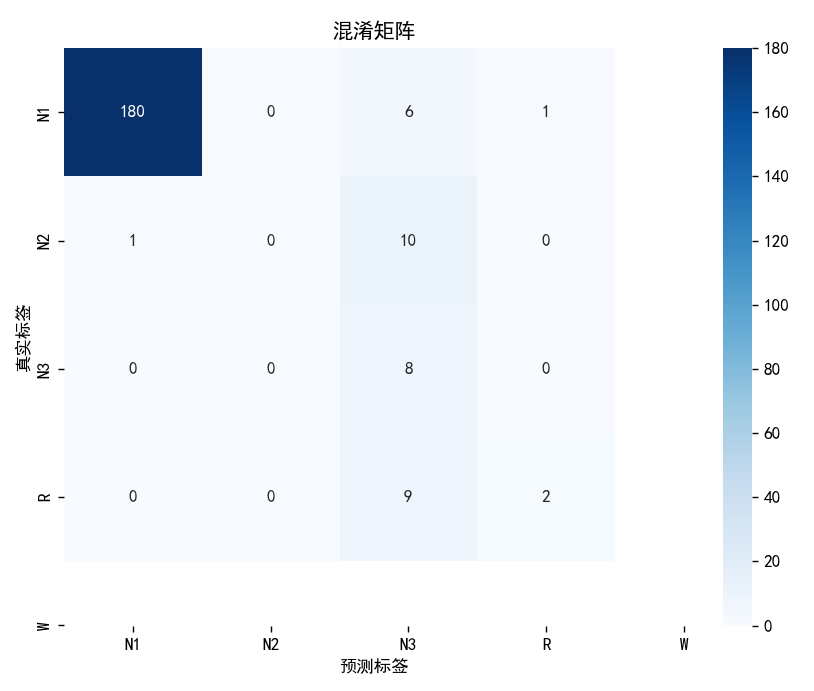
|  |
| --- |
|  |
| 这里是因为神经网络运算的时候里面的变量都是float32的格式，不需要用float64，float32的精度就足够了。而我给的是float64的格式，因此这里手动转为float32即可  在CNN模型的forward函数最开始加上  x = x.float()  # 将输入数据类型转换为 torch.float32, x.shape = (32,1,31)  即可解决 |

### 模型评价

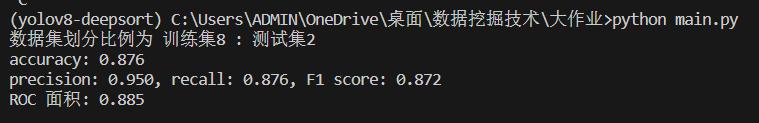
最终模型计算得到的各个评价指标如下。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| accuracy | precision | recall | F1 score | ROC\_auc |
| **0.876** | **0.950** | **0.876** | 0.872 | 0.885 |





### 运行结果截图



### 结论分析

综合来看，一维CNN深度学习模型在睡眠分期预测任务上表现良好。准确率、查准率、召回率、F1 分数和 ROC AUC Score 的高值表明模型对于不同类别的预测能力均较强。

但是这里使用的数据集还是数据量有些少了，模型的泛化性不能表现出来，后期可以尝试在其他更大的数据集上进行实验。

### 参考文献列表

[1] Sun C, Hong S, Wang J, et al. A systematic review of deep learning methods for modeling electrocardiograms during sleep[J]. Physiological Measurement, 2022.

[2] Wei R, Zhang X, Wang J, et al. The research of sleep staging based on single-lead electrocardiogram and deep neural network[J]. Biomedical engineering letters, 2018, 8: 87-93.

[3] Sridhar N, Shoeb A, Stephens P, et al. Deep learning for automated sleep staging using instantaneous heart rate[J]. NPJ digital medicine, 2020, 3(1): 106.

[4] Sun H, Ganglberger W, Panneerselvam E, et al. Sleep staging from electrocardiography and respiration with deep learning[J]. Sleep, 2020, 43(7): zsz306.

[5] [基于深度学习心电图（ECG）特征识别\_ecg深度学习-CSDN博客](https://blog.csdn.net/qq_45637779/article/details/106470002)

[6] [[CBM 2020] 心电信号ECG数据深度学习方法的机遇与挑战：一项系统综述 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/577922190)

[7] [(6 封私信 / 26 条消息) 深度学习适合心电信号分析吗？ - 知乎 (zhihu.com)](https://www.zhihu.com/question/39522532)

### 全部源代码

Main.py

import torch

import torch.nn as nn

import torch.optim as optim

import torch.backends.cudnn as cudnn

from torch.utils.data import TensorDataset, DataLoader

from sklearn.metrics import accuracy\_score, precision\_recall\_fscore\_support, roc\_curve, auc, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

from utils import \*

from model import SleepStageModel

# 在开始之前设置随机种子

torch.manual\_seed(2023)

cudnn.deterministic = True

cudnn.benchmark = False  # 设置为 False 可以确保一些特定操作的确定性

# plt画图的中文字体

plt.rcParams['font.sans-serif']=['SimHei']

# 读入数据，其中len(ECG\_data) = 4065000，len(sleep\_labels) = 1084

ECG\_data, sleep\_labels = get\_data()

# 数据预处理

ECG\_data = np.array(ECG\_data)

# ECG\_data = rawdata\_remove\_noise(ECG\_data)  # 变差，可能是因为信息丢失了一部分

# ECG\_data = rawdata\_normalize(ECG\_data)  # 不变，神经网络很强

# ECG\_data = rawdata\_z\_score\_normalize(ECG\_data)  # 变差，改变了特征特性

# 提取特征

features = get\_features(ECG\_data)

features = np.array(features)

# features = feature\_minmax\_normalize(features)

# features = feature\_z\_score\_normalize(features)

# 转成tensor，为深度学习做准备

ECG\_data = torch.tensor(ECG\_data.copy())

sleep\_labels = torch.tensor(sleep\_labels)

features = torch.tensor(features)

# print(ECG\_data.shape, sleep\_labels.shape)  # ECG\_data.shape = ([4065000])，sleep\_labels.shape = ([1084])

# print(features.shape)  # features.shape = ([1084, 31])

# 初始化模型和损失函数、优化器

input\_size = features.shape[1]  # 特征的数量

NUM\_CLASSES = 5  # 分类的类别数

model = SleepStageModel(input\_size, NUM\_CLASSES)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

# 假设 features 和 sleep\_labels 是张量

train\_size = int(0.8 \* len(features))

test\_size = len(features) - train\_size

print(f"数据集划分比例为 训练集{8} : 测试集{2}")

# 使用切片进行划分

train\_features = features[:train\_size]

train\_labels = sleep\_labels[:train\_size]

test\_features = features[train\_size:]

test\_labels = sleep\_labels[train\_size:]

# 创建 DataLoader

BATCH\_SIZE = 32

train\_dataset = TensorDataset(train\_features, train\_labels)

test\_dataset = TensorDataset(test\_features, test\_labels)

train\_dataloader = DataLoader(train\_dataset, batch\_size=BATCH\_SIZE, shuffle=True)

test\_dataloader = DataLoader(test\_dataset, batch\_size=BATCH\_SIZE, shuffle=False)

# 训练模型

NUM\_EPOCHS = 80

for epoch in range(NUM\_EPOCHS):

    model.train()

    for features, labels in train\_dataloader:

        optimizer.zero\_grad()

        outputs = model(features.unsqueeze(1))  # 添加1维作为通道维度

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

# 在测试集上评估模型

model.eval()

all\_labels = []

all\_predictions = []

with torch.no\_grad():

    for features, labels in test\_dataloader:

        outputs = model(features.unsqueeze(1))

        \_, predicted = torch.max(outputs.data, 1)

        all\_labels.extend(labels.cpu().numpy())

        all\_predictions.extend(predicted.cpu().numpy())

# 计算准确率

accuracy = accuracy\_score(all\_labels, all\_predictions)

print(f'accuracy: {accuracy:.3f}')

# 计算查准率、查全率和 F1 分数

precision, recall, f1, \_ = precision\_recall\_fscore\_support(all\_labels, all\_predictions, average='weighted', zero\_division=1)

print(f'precision: {precision:.3f}, recall: {recall:.3f}, F1 score: {f1:.3f}')

# 绘制 ROC 曲线

fpr, tpr, thresholds = roc\_curve(all\_labels, all\_predictions, pos\_label=1)

roc\_auc = auc(fpr, tpr)

print(f"ROC 面积: {roc\_auc:.3f}")

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC 曲线 (面积 = {roc\_auc:.3f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive')

plt.ylabel('Ture Positive')

plt.title('ROC曲线')

plt.legend(loc="lower right")

plt.show()

# 绘制混淆矩阵

cm = confusion\_matrix(all\_labels, all\_predictions)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['N1', 'N2', 'N3', 'R', 'W'], yticklabels=['N1', 'N2', 'N3', 'R', 'W'])

plt.xlabel('预测标签')

plt.ylabel('真实标签')

plt.title('混淆矩阵')

plt.show()

model.py

import torch.nn as nn

# CNN模型

class SleepStageModel(nn.Module):

    def \_\_init\_\_(self, input\_size, num\_classes):

        super(SleepStageModel, self).\_\_init\_\_()

        self.cnn1d = nn.Sequential(

            nn.Conv1d(in\_channels=1, out\_channels=256, kernel\_size=3, stride=1, padding=1),

            nn.ReLU(),

            nn.MaxPool1d(kernel\_size=2, stride=2),

            nn.Conv1d(in\_channels=256, out\_channels=512, kernel\_size=3, stride=1, padding=1),

            nn.ReLU(),

            nn.MaxPool1d(kernel\_size=2, stride=2),

            # nn.Conv1d(in\_channels=128, out\_channels=256, kernel\_size=3, stride=1, padding=1),

            # nn.ReLU(),

            # nn.MaxPool1d(kernel\_size=2, stride=2),

            # nn.Conv1d(in\_channels=256, out\_channels=512, kernel\_size=3, stride=1, padding=1),

            # nn.ReLU(),

            # nn.MaxPool1d(kernel\_size=2, stride=2),

        )

        self.fc = nn.Linear(512 \* (input\_size // 4), num\_classes)

    def forward(self, x):

        x = x.float()  # 将输入数据类型转换为 torch.float32, x.shape = (32,1,31)

        x = self.cnn1d(x)

        x = x.view(x.size(0), -1)  # 展平

        x = self.fc(x)

        return x

utils.py

import numpy as np

import torch

from scipy.stats import skew, kurtosis

from scipy import signal

import pywt

def get\_data(ecg\_data\_path = r'ECG.txt', label\_path = r'label\_profusion.txt'):

    """

    返回:

    ECG\_data:  len(ECG\_data) = 4065000 = 125 \* 30 \* 1084, 每个浮点数代表一个时间点上的心脏电压值。

    sleep\_labels:  len(sleep\_labels) = 1084, 代表ECG每个epoch对应的睡眠分期标签, 其中睡眠分期标签的映射如下":"w:0 ;N1:1; N2:2;N3:3;R:4;

    """

    ECG\_data = open(ecg\_data\_path).read()         #设立data列表变量，python 文件流，.read读文件

    ECG\_data = ECG\_data.split( )                         #以空格为分隔符，返回数值列表data

    ECG\_data = [float(s) for s in ECG\_data]              #将列表data中的数值强制转换为float类型

    # len(ECG\_data) = 4065000 = 125 \* 30 \* 1084

    sleep\_labels = open(label\_path).read()         #设立data列表变量，python 文件流，.read读文件

    sleep\_labels = sleep\_labels.split( )                         #以空格为分隔符，返回数值列表data

    sleep\_labels = [int(s) for s in sleep\_labels]              #将列表data中的数值强制转换为float类型

    # len(sleep\_labels) = 1084

    return ECG\_data, sleep\_labels

import numpy as np

from scipy import signal

def rawdata\_remove\_noise(ecg\_data, sampling\_rate=125):

    # 使用滤波器去除高频噪声

    nyquist = 0.5 \* sampling\_rate

    low = 0.25

    high = 100.0

    low\_cutoff = low / nyquist

    high\_cutoff = min(high, 0.499 \* sampling\_rate) / nyquist

    b, a = signal.butter(1, [low\_cutoff, high\_cutoff], btype='band')

    filtered\_ecg = signal.filtfilt(b, a, ecg\_data)

    return filtered\_ecg

def rawdata\_normalize(ecg\_data):

    # 归一化数据，将数据缩放到 [0, 1] 范围

    normalized\_ecg = (((ecg\_data - np.min(ecg\_data)) / (np.max(ecg\_data) - np.min(ecg\_data))) - 0.5) \* 2

    return normalized\_ecg

def rawdata\_z\_score\_normalize(ecg\_data):

    mean = np.mean(ecg\_data)

    std = np.std(ecg\_data)

    normalized\_ecg = (ecg\_data - mean) / std

    return normalized\_ecg

def feature\_minmax\_normalize(features):

    min\_vals = features.min(axis=0, keepdims=True)

    max\_vals = features.max(axis=0, keepdims=True)

    normalized\_features = (features - min\_vals) / (max\_vals - min\_vals)

    return normalized\_features

def feature\_z\_score\_normalize(features):

    mean\_vals = features.mean(axis=0, keepdims=True)

    std\_devs = features.std(axis=0, keepdims=True)

    normalized\_features = (features - mean\_vals) / std\_devs

    return normalized\_features

def calculate\_slope(x, y):

    N = len(x)

    sum\_xy = np.sum(x \* y)

    sum\_x = np.sum(x)

    sum\_y = np.sum(y)

    sum\_x\_squared = np.sum(x\*\*2)

    slope = (N \* sum\_xy - sum\_x \* sum\_y) / (N \* sum\_x\_squared - sum\_x\*\*2)

    return slope

def extract\_features(raw\_signal):

    features = []

    # 时域特征

    features.append(np.mean(raw\_signal))

    features.append(np.std(raw\_signal))

    features.append(np.min(raw\_signal))

    features.append(np.max(raw\_signal))

    features.append(calculate\_slope(np.array(list(range(1,125\*30+1,1)))/1000,raw\_signal))  # 斜度2

    # features.append((raw\_signal[-1] - raw\_signal[0]) / len(raw\_signal))  # 斜度

    features.append(np.sum(np.square(raw\_signal)))  # 能量

    # 频域特征

    f, Pxx = signal.periodogram(raw\_signal)

    features.append(np.sum(Pxx))  # 功率谱密度总和

    # 小波变换系数

    coeffs = pywt.wavedec(raw\_signal, 'db1', level=5)

    for c in coeffs:

        features.extend([np.mean(c), np.std(c), skew(c), kurtosis(c)])

    return features

def get\_features(ECG\_data):

    """ECG\_data's length is 4065000"""

    feature\_list = []

    # 对每个epoch提取特征

    for i in range(len(ECG\_data) // 125 // 30):

        feature\_list.append(extract\_features(ECG\_data[i\*125\*30:(i+1)\*125\*30]))

    return feature\_list