**基于离散小波变换的信号特征提取方法在心电数据集分类上的使用**

**摘 要**

在信号的特征提取与分类中，心电信号等频谱随时间变化的信号往往较难使用傅立叶变换相关的方法处理，而对心电信号进行分类又在医学领域的应用有着深远意义。

为此，本文用Python实现了一套基于离散小波变换（DWT）的特征提取算法，与“信号滤波后进行基于统计量进行特征提取”这一较常见的方法在心电图信号分类这一应用场景上相对比，并在MIT-BIH的ECG数据集上达到了90%以上的分类准确率。

**关键词：信号分类、信号滤波、信号特征提取、统计量、离散小波变换**

1. **引言**

在当今的大数据时代，心电信号（ECG）是数据科学中应用广泛的一类数据，它的识别与分类在医学诊断等领域有着较强意义。对心电信号进行分类，首要的环节就是特征提取。而心电信号不同于一些常见的简单信号，它的频谱随时间变化且它在时域上的长度较大（每个信号在时域上的样本点个数都为65536），这给一些常见的基于频域的信号特征提取方法（如快速傅立叶变换（FFT）、功率谱密度（PSD））造成了较大困难。

对于这种信号的分类有多种方法：UCI的HAR数据集[1]中提出了先对信号进行滤波、重叠窗口等处理后再进行统计学特征提取的信号特征提取方法，该方法在HAR数据集上实现了90%的分类准确率；参考资料[2]提出了利用FFT、PSD、自相关三种方法分别将时域信号转化为频域信号并选取前5个频域波峰作为信号特征的特征提取方法，在UCI HAR数据集上实现了88%的分类准确率；文献[3]中提出了短时傅立叶变换的方法，相比于FFT而言，这种方法在应用FFT之前，通过构建滑动窗口，将原始信号分为长度相等但可重叠的几个部分，如此可以确定信号的每个部分的特定频率。然而这种方法的局限性在于，当预先设定的滑动窗口的大小较小时，对各个时间段信号频率的了解更精确，但是对信号总体的频率值的了解就会较少，而当预先设定的滑动窗口的大小较大时，对总体频率值的了解会更多，但是对各个时间段的信号频率无法精确了解；文献[4]中提出的通过小波变换进行特征提取的思路则是一种更好的方法。小波变换根据小波本身的频率不同，在频域和时域都具有较好的识别性能。它能够体现出信号中存在的各种频率，以及这些频率值分别出现的时间。

总的来说，原始的时间序列信号在时域中可以很好地被识别与区分特征，而在频域中没有任何特征；傅立叶变换在频域中可以很好地被区分，而在时域中没有特征；短时傅立叶变换在时域和频域都具有中等大小的可被区分的性能；小波变换则要分以下两种情形讨论：对于较小的频率值，在频域中的分辨率高于在时域中的分辨率；对于较大的频率值，在时域中的分辨率高于在时域中的分辨率，因此，小波变换是一种较好的折中方案。

Python语言的小波变换PyWavelets包的功能仍在被不断完善，但许多统计学背景出身的数据科学家并不能很好地将这种方法应用到数据处理中，小波变换在数据处理、数据挖掘相关领域的文献也并不多见。因此，小波变换等数字信号处理的知识未来在数据科学的各个领域中将会有着更广泛的应用前景。

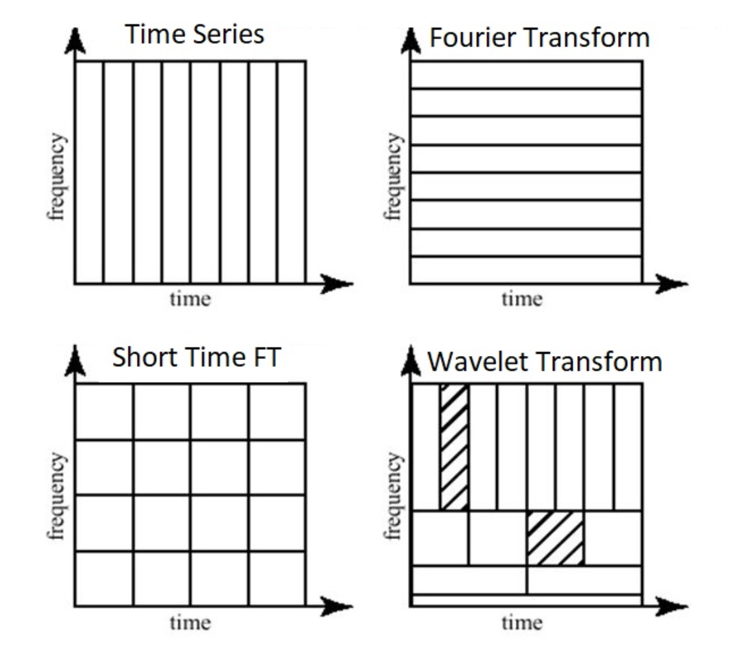


图1 不同变换在时域和频域上的分辨率

1. **算法原理与对比分析**

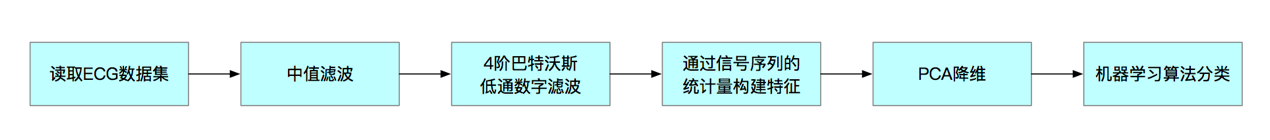
**2.1 UCI的HAR数据集提供的信号特征提取方法**

图2 UCI信号分类算法流程图

* + 1. **数据集的读取和预处理**

由于ECG数据集以MATLAB对应的数据格式存储，因此使用Python scipy包中的scipy.io.loadmat()函数打开此文件。注意到数据集文件是将信号数据和标签数据分开保存，因此我们将标签字符串转为数字后，通过两个单独的数组来保存这些数据。

在读取数据集后发现，MIT-BIH的ECG数据集并不平衡，其中有96个ARR（心率不齐）测量值，36个NSR（健康心率）测量值和30个CHF（充血性心力衰竭）测量值。为此，我们对数据集中ARR这一类的数据样本进行欠采样，从中随机抽取35个样本保留，并将其余的ARR样本移出数据集，由此得到了较为平衡的数据集。

* + 1. **信号的滤波**

上一步中读取的数据集分为信号和标签两部分，为了使信号的特征提取更加有效，采取中值滤波器和巴特沃斯滤波器依次对信号滤波。

中值滤波器是一种非线性的滤波器，它把信号序列中某点的值用该点周围的奇数个点的中位数代替，从而使信号变得更加平滑，并消除信号中一些孤立的噪声点。对于心电信号，它的干扰噪声较强，因此，本文首先采取窗口大小为5的中值滤波器（把每个点的值用该点的邻域中5个点的中值代替），对信号进行滤波，从而使信号的特征提取算法更加有效。

查阅资料[4]得知，心电信号的频率主要集中在5-20Hz，而大于30Hz的部分为肌电信号产生的干扰。根据心电信号的此种特性，应当选用一种截止频率为20-30Hz的低通滤波器对心电信号进行滤波。而巴特沃斯滤波器在通带上的频率响应曲线可以达到最大限度的平坦，且在阻带上趋近于零，因此选用巴特沃斯低通滤波器对信号进行进一步处理。

根据公式，设置截止频率为，且已知采样频率为时，归一化截止频率。滤波后的信号如下图所示：

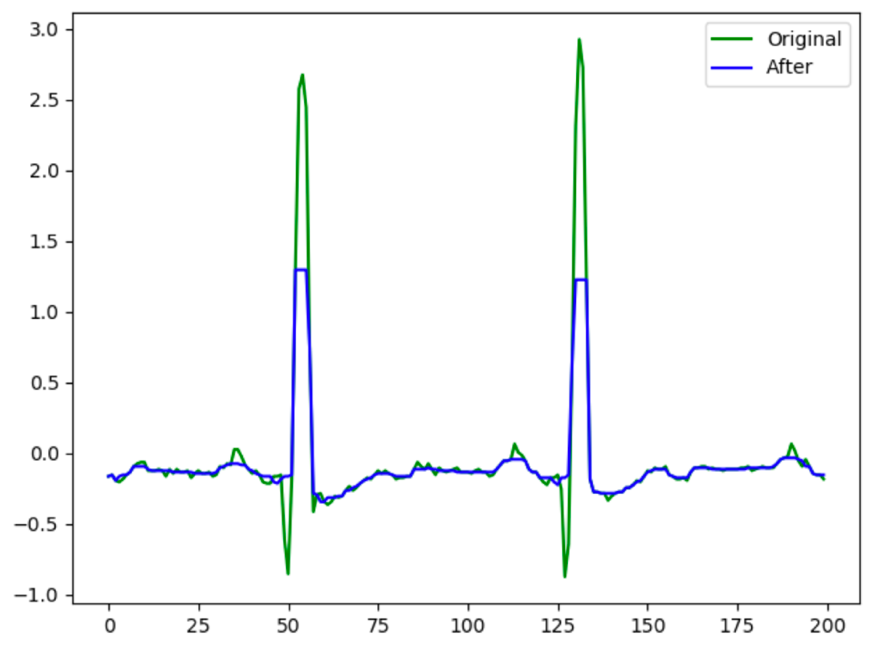


图3 滤波后的信号示例（取前200个样本点进行展示）

* + 1. **基于统计量的特征提取**

在对信号进行滤波后，尽可能多地选取一些时域、频域上的统计量特征作为预选特征。选取的特征如下表所示：

|  |  |
| --- | --- |
| 特征名称 | 定义 |
| min | 最小值 |
| max | 最大值 |
| per5 | 5%分位数 |
| per25 | 下四分位数 |
| median | 中位数 |
| per75 | 上四分位数 |
| per95 | 95%分位数 |
| mean | 均值 |
| std | 标准差 |
| skew | 时域偏度 |
| kurtosis | 时域峰度 |
| pass\_zero | 过零次数（时域信号有多少次穿过y轴） |
| pass\_mean | 过均值次数（时域信号有多少次穿过y=mean） |
| wskew | 频域偏度 |
| wkurtosis | 频域峰度 |
| entropy | 香农熵 |

将所有这些特征合并为一个数组，作为一个信号样本的特征，再经过PCA算法降维，选取其中分类效果较好的9种特征，放入分类器进行训练与测试。

上述算法在常见分类器上的准确率测试结果如下图，可以看出，这种直接提取统计量的特征提取方法在心电图信号的处理上效果并不理想，给出的最高分类准确率仅有77%。（图3滤波效果与图4结果可以在文末代码statistics.py中复现）

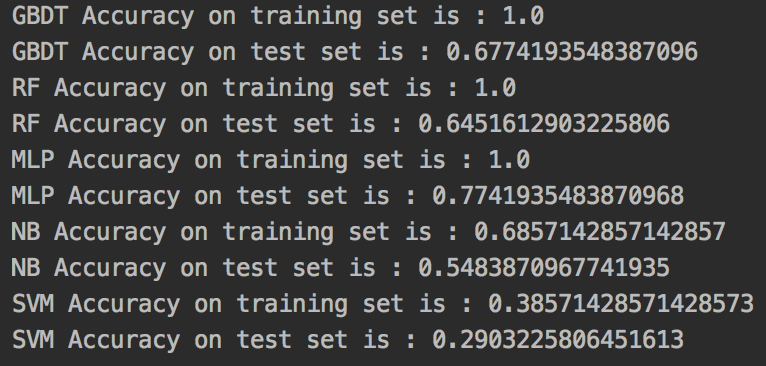


图4 UCI HAR直接提取统计量的特征提取算法准确率测试结果

* 1. **基于离散小波变换的信号特征提取方法**
     1. **数据集的读取和预处理**

本节中采用的数据集与2.1相同，数据集的读取和预处理过程与2.1.1中所述一致。

* + 1. **关于离散小波变换**

小波是均值为零且幅度迅速减小的振动波，它存在的时间是有限的。相比傅立叶变换而言，小波变换在频域随时间变化而变化的信号上有着更好的分辨率。连续小波变换和离散小波变换都基于小波的缩放和转移进行。其中小波的缩放是随时间增加或减少信号的过程，可以用公式表示，其中是正比例因子，与频率成反比关系；较大的比例因子会产生对应较低频率的小波，较小的比例因子会产生对应较高频率的小波。小波的转移则是根据信号长度延迟或提前小波的开始，用公式表示。

离散小波变换取决于如何离散化缩放比例和转换参数。一般离散小波变换中的缩放通过离散化的2的幂次得到，缩放比例可以用公式 表示，离散小波变换中的转移则是以缩放的整数倍发生，可以用公式表示。

为了将DWT应用于ECG数据集的信号，首先从最小的比例因子开始，它也对应着最高的频率，这意味着正在分析信号的最高频部分；第二步，比例因子增加2倍，对应地，频率也减少2倍，被分析的部分变为最大频率左右的部分；第三步，比例因子 增加4倍，被分析的部分变为最大频率左右的部分，如此不断分解下去，直到达到预设的最大分解级别。

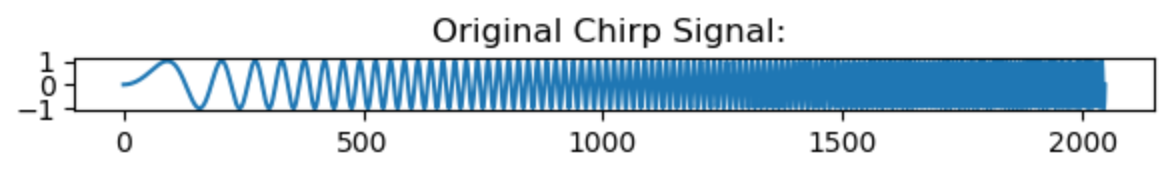
图5.1和5.2以单个信号为例，对DWT过程进行展示。（图片生成的代码见附录figure.py）图6为DWT过程的流程图。

图5.1 一个频域随时间大幅变化的信号

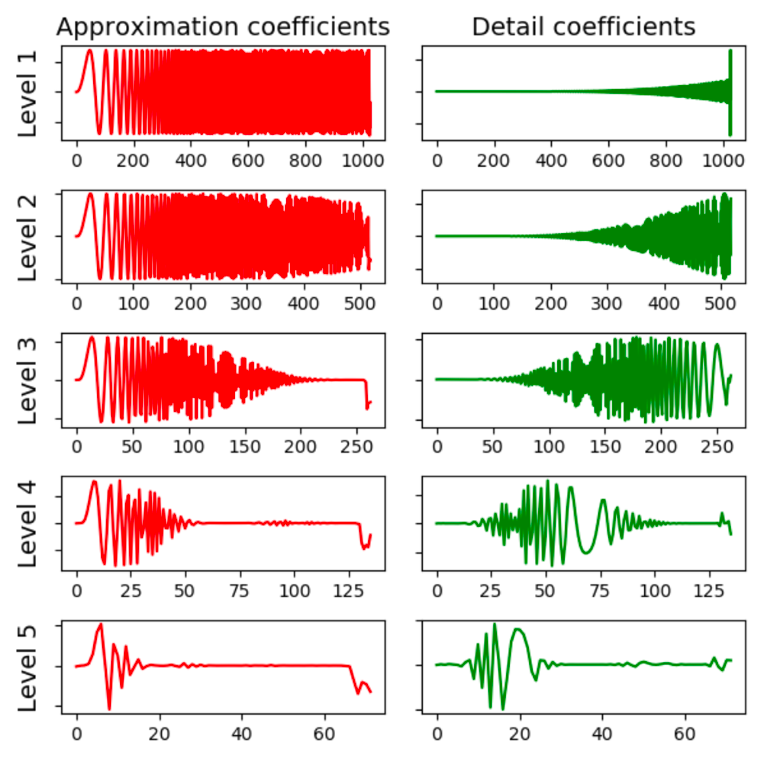


图5.2 上述信号在每一次重复DWT过程后的部分，右侧绿色信号为原信号中正在被分析的部分，左侧为剩余部分。

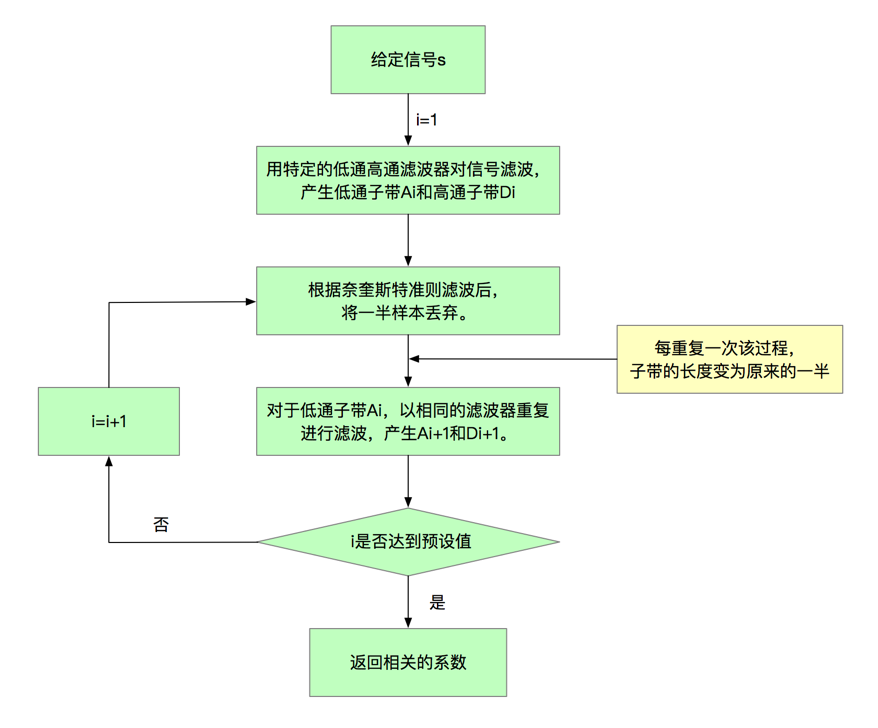


图6 DWT过程的流程图

在上述过程中，记预设值为，则最后产生的子带为。

* + 1. **基于离散小波变换的信号特征提取算法**

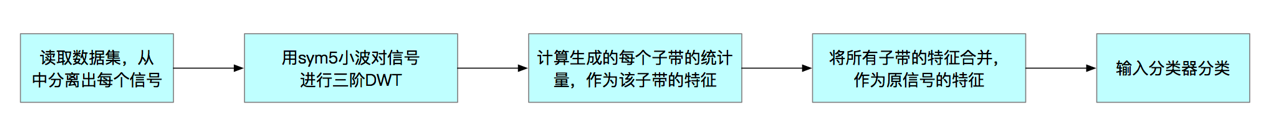
****

图7 基于DWT的特征提取算法的流程图

考虑到小波变换生成的多个子带可以更好地反映原信号总体的特征，因此，我们对每个子带进行与2.1.3中相同的特征提取，并输入分类器进行训练，经过选用Daubechies、Symlets、Coiflets等不同离散小波和不同的DWT阶数进行多次实验，发现sym5小波（即Symlets小波）做三阶DWT特征提取的效果最好，结合常用的随机森林分类器，在ECG信号的分类上可以达到95%的准确率。由此可见，2.2介绍的特征提取方法在心电信号分类上的效果要远远优于2.1提出的特征提取方法。（下图结果可由附录中wavelet.py复现）

图8基于DWT的特征提取算法的分类结果

1. **项目成果与结论**

本文结合[2]实现了一种用离散小波变换生成子带，对子带进行统计学特征提取后合并生成新特征的特征提取算法，并将它与UCI HAR数据集的说明文档中提出的信号特征提取算法进行对比。通过实验得知，前者在心电信号这种频域随时间变化的信号特征提取与分类上达到了95%的准确率，这要显著优于后者77%的准确率，在复杂信号的特征提取上有着较强的应用意义。

1. **总结与展望**
   1. **研究总结**

本文结合信号处理课程中学习到的一些关于傅立叶变换和滤波器的基础知识，对心电信号进行处理，将两种信号的特征提取算法在MIT-BIH的ECG数据集上进行分类准确率测试从而对比两者的性能。本文的主要结论如下：

对于一些常见的简单信号，利用文献[1]给出的特征提取方法或是利用文献[2]提出的基于FFT的特征提取方法，可以达到较高的准确率，但是对于频域随着时间有明显变化的信号，如心电信号，单纯的统计学特征提取或是FFT不能很好地提取出能够充分反映信号本身的特征。而基于DWT的一套特征提取方法，在心电信号、脑电信号等复杂信号的特征提取中，充分考虑了每个子带的性质，进而能够更好地反映整个信号的特征。这种方法在今后的学习生活中涉及到复杂信号，如心电、脑电、声音信号处理相关的问题，

* 1. **未来展望**

本文实现的基于离散小波变换的特征提取方法在心电数据集上有着较好的效果，但从信号处理的知识来看，短时傅立叶变换和连续小波变换也不失为一种好的协调时域和频域特征的方法。今后在心率、声音等一维信号相关的应用场景中，可以用类似的特征提取方法，对比短时傅立叶变换、CWT、DWT的准确率，并选择性能最好的方法使用。

1. **致谢**

感谢严老师这半年来在信号处理这门课程上对我们的教导，这让统计学背景出身的我们进一步拓宽了知识面，也在数据科学的各个应用领域（例如本文提出的复杂信号分类和特征提取在心电信号中的应用）上更进一步。

1. **参考文献**

[1] UCI HAR数据集<https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones>

[2]<http://ataspinar.com/2018/04/04/machine-learning-with-signal-processing-techniques/>

[3]张东,吴晓琳.短时傅立叶变换在振动信号处理中的应用[J].计算机与数字工程,2011,39(08):154-155+195.

[4]杨帮华,颜国正,鄢波.基于离散小波变换提取脑机接口中脑电特征[J].中国生物医学工程学报,2006(05):518-522.

[5]基于MATLAB的心电信号预处理<https://blog.csdn.net/zhaomengszu/article/details/78842613>

1. **附录**

Statistics.py

1. **import** scipy.io as sio
2. **from** collections **import** defaultdict
3. **from** sklearn.decomposition **import** PCA
4. **from** sklearn.svm **import** SVC
5. **from** sklearn.ensemble **import** RandomForestClassifier,GradientBoostingClassifier
6. **from** sklearn.neural\_network **import** MLPClassifier
7. **from** sklearn.naive\_bayes **import** GaussianNB
8. **import** numpy as np
9. **from** scipy **import** stats
10. **from** scipy.signal **import** filtfilt, butter
11. **import** matplotlib.pyplot as plt

14. **class** getFeature():
15. **def** calculate\_features(self, inputs):
16. inputs = np.array(inputs)
17. # 最小值
18. min = np.min(inputs)
19. # 最大值
20. max = np.max(inputs)
21. # 5%分位数
22. per5=np.nanpercentile(inputs, 5)
23. # 95%分位数
24. per95=np.nanpercentile(inputs, 95)
25. # 上四分位数
26. per25 = np.nanpercentile(inputs, 25)
27. # 下四分位数
28. per75 = np.nanpercentile(inputs, 75)
29. # 均值
30. mean = np.mean(inputs)
31. # 中值
32. median = np.median(inputs)
33. # 中值绝对偏差
34. mad = stats.median\_absolute\_deviation(inputs)
35. # 标准差
36. std = np.std(inputs, ddof=1)
37. # 偏度
38. skew = stats.skew(inputs)
39. # 峰度
40. kurtosis = stats.kurtosis(inputs)
41. # 四分位数范围
42. iqr = stats.iqr(inputs)
43. # 过零率
44. cross\_zero = np.nonzero(np.diff(np.array(inputs) > 0))[0]
45. pass\_zero=len(cross\_zero)/len(inputs)\*10
46. # 过均值率
47. cross\_mean = np.nonzero(np.diff(np.array(inputs) > np.nanmean(inputs)))[0]
48. pass\_mean=len(cross\_mean)/len(inputs)\*10
49. # 频域偏度系数
50. wskew = stats.skew(inputs)
51. # 频域峰度系数
52. wkurtosis = stats.kurtosis(inputs)
53. # 将所有特征合并为数组
54. array = [min, max, mean, median, mad, std, skew, kurtosis, iqr, per5, per25, per75, per95, pass\_zero, pass\_mean, wskew, wkurtosis]
55. **return** array

58. **def** get\_train\_test(df, y\_col, x\_cols, ratio):  # 将数据按比例切分为训练集、测试集
59. region = np.random.rand(len(df)) < ratio
60. df\_train = df[region]
61. df\_test = df[~region]
62. Y\_train = df\_train[y\_col].values
63. Y\_test = df\_test[y\_col].values
64. X\_train = df\_train[x\_cols].values
65. X\_test = df\_test[x\_cols].values
66. **return** df\_train, df\_test, X\_train, Y\_train, X\_test, Y\_test

69. **def** medfit(vector, n\_odd):  # 中值滤波
70. odd=int((n\_odd-1)/2)
71. fitted=[0]\*len(vector)
72. **for** i **in** range(0,odd):
73. fitted[i]=vector[i]
74. **for** i **in** range(len(vector)-odd,len(vector)):
75. fitted[i]=vector[i]
76. **for** i **in** range(odd,len(vector)-odd):
77. obj=vector[i-odd:i+odd+1]
78. obj=sorted(obj)
79. fitted[i]=obj[odd]
80. **return** fitted

83. **def** butterworth(vector, N, Wn):
84. b, a = butter(N, Wn, btype='lowpass', output='ba')
85. arr = filtfilt(b, a, vector)
86. **return** arr

89. # 读取数据（这里原数据集保存在.mat文件中，用scipy.sio.loadmat读取）
90. filename = './balanced\_data.mat'
91. ecg\_data = sio.loadmat(filename)  # ecg\_data中包括信号和标签两部分
92. ecg\_signals = ecg\_data['ECGData'][0][0][0]  # ecg\_data中的信号
93. ecg\_labels\_ = ecg\_data['ECGData'][0][0][1]  # ecg—data中的标签
94. ecg\_labels = list(map(**lambda** x: x[0][0], ecg\_labels\_))  # 将label提取出来作为输入分类器的标签
95. # print(ecg\_labels)
96. dict\_ecg\_data = defaultdict(list)  # 构建一个字典，将信号和标签对应起来
97. **for** ii, label **in** enumerate(ecg\_labels):  # ii为序号，label为标签值
98. dict\_ecg\_data[label].append(ecg\_signals[ii])
100. list\_labels = []
101. list\_features = []
102. raw\_data = []
103. # 把ecg数据集转为一个矩阵，每行是一个信号及其标签。
104. **for** label, signals **in** dict\_ecg\_data.items():
105. y\_value = list(dict\_ecg\_data.keys()).index(label)  # 把y的字符串标签转换为数字1-3
106. plot=signals[0]
107. x=np.linspace(0,199,num=200)
108. y=signals[0][0:200]
109. plt.plot(x, y, "g-", label="Original")
110. plot=medfit(plot,5)
111. plot1=butterworth(plot, 8, 1/9)
112. y2=plot[0:200]
113. plt.plot(x, y2, "b-", label="After")
114. plt.legend()
115. plt.show()
116. **for** signal **in** signals:  # v中是同一种标签对应的所有信号，因此signal in v代表同种类型的所有信号。signal代表单个信号。
117. signa=signal.tolist()
118. sign=medfit(signa,5)  # 中值滤波
119. sig=butterworth(sign, 4, 1/9)  # 四阶巴特沃斯低通数字滤波器
120. si=sig.tolist()
121. si.append(y\_value)
122. raw\_data.append(si)
123. # PCA降维，确定5个最优的统计量作为特征
124. FEA=getFeature()
125. final\_data=[]
126. **for** line **in** raw\_data:
127. features=FEA.calculate\_features(line)
128. features.append(line[-1])
129. final\_data.append(features)
130. X,Y=list(),list()
131. **for** i **in** range(int(len(final\_data))):
132. X.append(final\_data[i][0:len(final\_data[i])-1])
133. Y.append(final\_data[i][-1])
134. pca=PCA(n\_components=9)
135. new\_data=pca.fit\_transform(X)
136. together=[]
137. **for** i **in** range(len(new\_data)):
138. a=new\_data[i].tolist()
139. a.append(Y[i])
140. together.append(a)
141. np.random.seed(1)
142. np.random.shuffle(together)
143. X\_train,Y\_train=list(),list()
144. X\_test,Y\_test=list(),list()
145. **for** i **in** range(int(len(together)\*0.7)):
146. X\_train.append(together[i][0:len(together[i])-1])
147. Y\_train.append(together[i][-1])
148. **for** i **in** range(int(len(together)\*0.7),len(together)):
149. X\_test.append(together[i][0:len(together[i])-1])
150. Y\_test.append(together[i][-1])
152. clf=GradientBoostingClassifier(n\_estimators=1000)
153. clf.fit(X\_train, Y\_train)
154. **print**("GBDT Accuracy on training set is : {}".format(clf.score(X\_train, Y\_train)))
155. **print**("GBDT Accuracy on test set is : {}".format(clf.score(X\_test, Y\_test)))
156. clf=RandomForestClassifier(n\_estimators=1000)
157. clf.fit(X\_train, Y\_train)
158. **print**("RF Accuracy on training set is : {}".format(clf.score(X\_train, Y\_train)))
159. **print**("RF Accuracy on test set is : {}".format(clf.score(X\_test, Y\_test)))
160. # print(classification\_report(Y\_test, Y\_test\_pred))
161. clf=MLPClassifier(max\_iter=2000)
162. clf.fit(X\_train, Y\_train)
163. **print**("MLP Accuracy on training set is : {}".format(clf.score(X\_train, Y\_train)))
164. **print**("MLP Accuracy on test set is : {}".format(clf.score(X\_test, Y\_test)))
166. clf = GaussianNB()
167. clf = clf.fit(X\_train, Y\_train)
168. **print**("NB Accuracy on training set is : {}".format(clf.score(X\_train, Y\_train)))
169. **print**("NB Accuracy on test set is : {}".format(clf.score(X\_test, Y\_test)))
171. svm = SVC()
172. svm = svm.fit(X\_train,Y\_train)
173. **print**("SVM Accuracy on training set is : {}".format(svm.score(X\_train, Y\_train)))
174. **print**("SVM Accuracy on test set is : {}".format(svm.score(X\_test, Y\_test)))

Wavelet.py

1. **import** numpy as np
2. **import** pandas as pd
3. **import** scipy.io as sio
5. **import** pywt
6. **import** scipy.stats
8. **from** collections **import** defaultdict, Counter
10. **from** sklearn.ensemble **import** GradientBoostingClassifier
11. **from** sklearn.ensemble **import** RandomForestClassifier
12. **from** sklearn.neural\_network **import** MLPClassifier
13. **from** sklearn.svm **import** SVC
14. **from** sklearn.naive\_bayes **import** GaussianNB
16. # 数据集中有35个ARR（心率失常），30个CHF（心力衰竭），36个NSR（窦性心率正常）

19. **class** Wavelet(object):
20. **def** \_\_init\_\_(self, array):
21. self.array = array
23. **def** \_\_del\_\_(self):
24. class\_name = self.\_\_class\_\_.\_\_name\_\_
26. **def** get\_entropy(self):  # 通过概率值计算熵，作为信号复杂度的度量
27. counter\_set = Counter(self.array).most\_common()
28. # collection包中Counter函数的作用：把每个元素及其出现的次数进行统计。
29. # 例: Counter('daddy')输出为Counter({'d':3, 'a':1, 'y':1})
30. # Counter.most\_common则是把Counter的输出结果转化为数组，且将各个数出现的顺序按照"出现次数从多到少"排列。
31. possibilities = []
32. **for** item **in** counter\_set:
33. possibilities.append(item[1]/len(self.array))
34. entropy = scipy.stats.entropy(possibilities)
35. **return** entropy
37. **def** common\_statistics(self):  # 获取小波变换后信号几个常见的统计量
38. # np.nanpercentile计算百分比分位数。
39. per5 = np.nanpercentile(self.array, 5)  # 5%分位
40. per25 = np.nanpercentile(self.array, 25)
41. per75 = np.nanpercentile(self.array, 75)
42. per95 = np.nanpercentile(self.array, 95)
43. per50 = np.nanpercentile(self.array, 50)  # 中位数
44. mean = np.nanmean(self.array)
45. std = np.nanstd(self.array)
46. var = np.nanvar(self.array)
47. rms = np.nanmean(np.sqrt(self.array\*\*2))  # 均方根
48. **return** [per5, per25, per75, per95, per50, mean, std, var, rms]
50. **def** cross\_times(self):
51. cross\_zero = np.nonzero(np.diff(np.array(self.array) > 0))[0]
52. # np.diff对数组做差分(True和False差分时视为0，1)，np.nonzero取出数组非零值的下标
53. zero\_cross\_time = len(cross\_zero)
54. # 过零次数，即信号穿过y轴的次数
55. cross\_mean = np.nonzero(np.diff(np.array(self.array) > np.nanmean(self.array)))[0]
56. mean\_cross\_time = len(cross\_mean)
57. # 过均值次数，即信号穿越y=平均值这一直线的次数
58. **return** [zero\_cross\_time, mean\_cross\_time]
60. **def** get\_features(self):
61. ent = self.get\_entropy()
62. cross = self.cross\_times()
63. stat = self.common\_statistics()
64. **return** [ent] + cross + stat
65. # 特征：方差、标准差、平均值、中位数、下四分位数、上四分位数、5%分位数、95%分位数、均方根、熵、过零次数、过均值次数

68. # =====================================================三种方法公共部分=====================================================
69. **def** get\_train\_test(df, y\_col, x\_cols, ratio):  # 将数据按比例切分为训练集、测试集
70. np.random.seed(0)  # 随机种子
71. region = np.random.rand(len(df)) < ratio
72. df\_train = df[region]
73. df\_test = df[~region]
74. Y\_train = df\_train[y\_col].values
75. Y\_test = df\_test[y\_col].values
76. X\_train = df\_train[x\_cols].values
77. X\_test = df\_test[x\_cols].values
78. **return** df\_train, df\_test, X\_train, Y\_train, X\_test, Y\_test

81. # 读取数据（这里原数据集保存在.mat文件中，用scipy.sio.loadmat读取）
82. filename = './balanced\_data.mat'
83. ecg\_data = sio.loadmat(filename)  # ecg\_data中包括信号和标签两部分
84. ecg\_signals = ecg\_data['ECGData'][0][0][0]  # ecg\_data中的信号
85. ecg\_labels\_ = ecg\_data['ECGData'][0][0][1]  # ecg—data中的标签
86. ecg\_labels = list(map(**lambda** x: x[0][0], ecg\_labels\_))  # 将label提取出来作为输入分类器的标签
87. # print(ecg\_labels)
88. dict\_ecg\_data = defaultdict(list)  # 构建一个字典，将信号和标签对应起来
89. **for** ii, label **in** enumerate(ecg\_labels):  # ii为序号，label为标签值
90. dict\_ecg\_data[label].append(ecg\_signals[ii])
92. list\_labels = []
93. list\_features = []
94. **for** label, signals **in** dict\_ecg\_data.items():
95. y\_value = list(dict\_ecg\_data.keys()).index(label)  # 把y的字符串标签转换为数字1-3
96. **for** signal **in** signals:  # v中是同一种标签对应的所有信号，因此signal in v代表同种类型的所有信号。signal代表单个信号。
97. features = []
98. list\_labels.append(y\_value)
99. #print('%%%',signal)
100. a=3
101. list\_wavelet = pywt.wavedec(signal, 'sym5', level=a)  # 用sym5小波实现3阶DWT
102. # list\_wavelet是wavedec函数返回的系数数组，
103. **for** list\_value **in** list\_wavelet:  # 对每个子带执行以下操作
104. Wave = Wavelet(list\_value)  # 从类Wavelet中实现特征提取
105. features += Wave.get\_features()
106. **del** Wave  # 销毁对象
107. list\_features.append(features)
108. df = pd.DataFrame(list\_features)
109. ycol = 'y'
110. xcols = list(range(df.shape[1]))
111. df.loc[:,ycol] = list\_labels
112. df\_train, df\_test, X\_train, Y\_train, X\_test, Y\_test = get\_train\_test(df, ycol, xcols, ratio = 0.7)
113. cls = GradientBoostingClassifier(n\_estimators=2000)
114. cls.fit(X\_train, Y\_train)
115. train\_score1 = cls.score(X\_train, Y\_train)
116. test\_score1 = cls.score(X\_test, Y\_test)
117. **print**("When a = {}, The Train Score of GBDT is {}".format(a,train\_score1))
118. **print**("When a = {}, The Test Score of GBDT is {}".format(a,test\_score1))
119. clf = RandomForestClassifier(n\_estimators=2000)
120. clf.fit(X\_train, Y\_train)
121. train\_score2 = clf.score(X\_train, Y\_train)
122. test\_score2 = clf.score(X\_test, Y\_test)
123. **print**("When a = {}, The Train Score of RF is {}".format(a,train\_score2))
124. **print**("When a = {}, The Test Score of RF is {}".format(a,test\_score2))
125. mlp = MLPClassifier(max\_iter=1000)
126. mlp.fit(X\_train, Y\_train)
127. train\_score3 = mlp.score(X\_train, Y\_train)
128. test\_score3 = mlp.score(X\_test, Y\_test)
129. **print**("When a = {}, The Train Score of MLP is {}".format(a,train\_score3))
130. **print**("When a = {}, The Test Score of MLP is {}".format(a,test\_score3))
131. svm = SVC()
132. svm.fit(X\_train, Y\_train)
133. train\_score4 = svm.score(X\_train, Y\_train)
134. test\_score4 = svm.score(X\_test, Y\_test)
135. **print**("When a = {}, The Train Score of SVM is {}".format(a,train\_score4))
136. **print**("When a = {}, The Test Score of SVM is {}".format(a,test\_score4))
137. nb = GaussianNB()
138. nb.fit(X\_train, Y\_train)
139. train\_score5 = nb.score(X\_train, Y\_train)
140. test\_score5 = nb.score(X\_test, Y\_test)
141. **print**("When a = {}, The Train Score of Naive Bayes is {}".format(a,train\_score5))
142. **print**("When a = {}, The Test Score of Naive Bayes is {}".format(a,test\_score5))
143. scores=[test\_score1,test\_score2,test\_score3,test\_score4,test\_score5]
144. max\_score=max(scores)
145. **print**("Symlets小波变换分解阶数为{}阶时，常见分类器最高准确率为{}".format(a,max\_score))

Figure.py

1. **import** pywt
2. **import** numpy as np
3. **import** matplotlib.pyplot as plt
5. x = np.linspace(0, 1, num=2048)
6. chirp\_signal = np.sin(250 \* np.pi \* x \*\* 2)
8. fig, ax = plt.subplots(figsize=(6, 1))
9. ax.set\_title("Original Chirp Signal: ")
10. ax.plot(chirp\_signal)
11. plt.show()
13. data = chirp\_signal
14. waveletname = 'sym5'
16. fig, axarr = plt.subplots(nrows=5, ncols=2, figsize=(6, 6))
17. **for** ii **in** range(5):
18. (data, coeff\_d) = pywt.dwt(data, waveletname)
19. axarr[ii, 0].plot(data, 'r')
20. axarr[ii, 1].plot(coeff\_d, 'g')
21. axarr[ii, 0].set\_ylabel("Level {}".format(ii + 1), fontsize=14, rotation=90)
22. axarr[ii, 0].set\_yticklabels([])
23. **if** ii == 0:
24. axarr[ii, 0].set\_title("Approximation coefficients", fontsize=14)
25. axarr[ii, 1].set\_title("Detail coefficients", fontsize=14)
26. axarr[ii, 1].set\_yticklabels([])
27. plt.tight\_layout()
28. plt.show()