

心跳信号分类预测

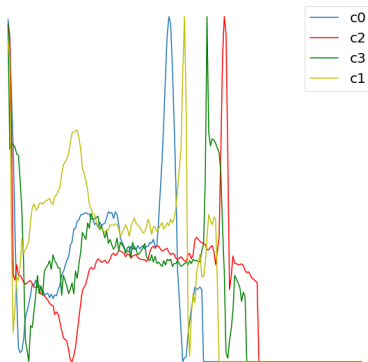
李阳

山东大学计算机科学与技术学院
2019 级智能班

2021 年 5 月 26 日

问题分析

- ① 多分类问题
- ② 数据量大
- ③ 特征是一串心跳序列
- ④ 结果提交的是 4 种不同心跳信号预测的概率，而非单一的预测所属分类



数据查看

- ① 实际是多个特征伪装成的一个特征
- ② 无匿名特征（与贷款违约预测相比）
- ③ 无缺省值
- ④ 特征之间关联性较大，主要是时序上的关联

| | id | heartbeat_signals | label |
|-------|-------|---|-------|
| 0 | 0 | 0.9912297987616655,0.9435330436439665,0.764677... | 0.0 |
| 1 | 1 | 0.9714822034884503,0.9289687459588268,0.572932... | 0.0 |
| 2 | 2 | 1.0,0.9591487564065292,0.7013782792997189,0.23... | 2.0 |
| 3 | 3 | 0.9757952826275774,0.9340884687738161,0.659636... | 0.0 |
| 4 | 4 | 0.0,0.055816398940721094,0.26129357194994196,0... | 2.0 |
| 99995 | 99995 | 1.0,0.677705342021188,0.22239242747868546,0.25... | 0.0 |
| 99996 | 99996 | 0.9268571578157265,0.9063471198026871,0.636993... | 2.0 |
| 99997 | 99997 | 0.9258351628306013,0.5873839035878395,0.633226... | 3.0 |
| 99998 | 99998 | 1.0,0.9947621698382489,0.8297017704865509,0.45... | 2.0 |
| 99999 | 99999 | 0.9259994004527861,0.916476635326053,0.4042900... | 0.0 |

数据处理

- ① 通过替换数据类型、用 category 类代替 object 类的方法来减小内存占用
- ② 将 heartbeat_signals 列分割成小的特征
- ③ 观察到 id 列无缺省值无异常值，所以在模型拟合时可以去掉，直接用 DataFrame 的默认索引代替 id 值

| | id | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 | ... | t196 | t197 | t198 | t199 | t200 | t201 | t202 | t203 | t204 | label |
|-------|---------|----------|----------|----------|----------|----------|----------|----------|----------|----------|-----|------|------|------|------|------|------|------|------|------|-------|
| 0 | 0.0 | 0.991211 | 0.943359 | 0.764648 | 0.618652 | 0.379639 | 0.198796 | 0.040222 | 0.026001 | 0.031708 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1 | 1.0 | 0.971680 | 0.929199 | 0.572754 | 0.178467 | 0.122986 | 0.132324 | 0.094421 | 0.089600 | 0.030487 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2 | 2.0 | 1.000000 | 0.958984 | 0.701172 | 0.231812 | 0.000000 | 0.080688 | 0.128418 | 0.187500 | 0.280762 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 |
| 3 | 3.0 | 0.975586 | 0.934082 | 0.659668 | 0.249878 | 0.237061 | 0.281494 | 0.249878 | 0.249878 | 0.241455 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 4.0 | 0.000000 | 0.055817 | 0.261230 | 0.359863 | 0.433105 | 0.453613 | 0.499023 | 0.542969 | 0.616699 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 |
| 99995 | 99995.0 | 1.000000 | 0.677734 | 0.222412 | 0.257080 | 0.204712 | 0.054657 | 0.026154 | 0.118164 | 0.244873 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 99996 | 99996.0 | 0.926758 | 0.906250 | 0.637207 | 0.415039 | 0.374756 | 0.382568 | 0.358887 | 0.341309 | 0.336426 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 |
| 99997 | 99997.0 | 0.925781 | 0.587402 | 0.633301 | 0.632324 | 0.639160 | 0.614258 | 0.599121 | 0.517578 | 0.403809 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 |
| 99998 | 99998.0 | 1.000000 | 0.994629 | 0.829590 | 0.458252 | 0.264160 | 0.240234 | 0.213745 | 0.189331 | 0.203857 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 |
| 99999 | 99999.0 | 0.925781 | 0.916504 | 0.404297 | 0.000000 | 0.262939 | 0.385498 | 0.361084 | 0.332764 | 0.339844 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

方法选用

主要考虑到三种方法

- AdaBoost
- 元分类器是支持向量机 (SVM) 的 AdaBoost
- 随机森林 Random Forest

树形结构的算法大多不需要进行降维处理，可以保持特征的可解释性

AdaBoost

- ① 最常见的 AdaBoost 元分类器是决策树，sklearn 中默认使用一层的决策树来作为它的元分类器
- ② 尝试以不同层数的决策树作为元分类器，精确度先增加后减少

```
层数>1 精确度均值: 0.650 精确度方差: 0.059
层数>2 精确度均值: 0.767 精确度方差: 0.047
层数>3 精确度均值: 0.873 精确度方差: 0.027
层数>4 精确度均值: 0.926 精确度方差: 0.018
层数>5 精确度均值: 0.938 精确度方差: 0.013
层数>6 精确度均值: 0.947 精确度方差: 0.010
层数>7 精确度均值: 0.947 精确度方差: 0.013
层数>8 精确度均值: 0.951 精确度方差: 0.013
层数>9 精确度均值: 0.951 精确度方差: 0.014
层数>10 精确度均值: 0.950 精确度方差: 0.012
```

AdaBoost + SVM

AdaBoost 接受所有能样本赋权的分类器作为其元分类器

- 最常见的赋权分类器有两种，即决策树和支持向量机
- 将 AdaBoost 和 SVM 结合，尝试找到一个较优的元分类器
- 实际效果和一层决策树的 AdaBoost 一致，并不是很优

准确率均值: 0.648 方差: 0.002

对比之下，还是决策树更适合与 AdaBoost 搭配

Random Forest

拟合效果：

准确率均值：0.946 方差：0.010

随机森林的优势

- ① 随机森林适用于拥有大型数据集的情况，可以处理数千个输入变量而无需变量删除
- ② 能够处理很高维度的数据，并且不用做特征选择（因为特征子集是随机选择的）
- ③ 可以在大部分数据丢失时保持准确性
- ④ 模型泛化能力强
- ⑤ 对于不平衡的数据集来说，它可以平衡误差
- ⑥ 不需要很多参数调整就可以达到不错的效果

选取最优方案

可以看出以 8 层决策树为元分类器的 AdaBoost 和随机森林的准确率较高，表现较好，最终选择使用随机森林

进一步优化

- ① PCA 降维，减少随机森林要处理的特征数量，加快随机森林模型的训练速度
- ② 对模型进行超参数调优，进一步提高准确率

计算成本高是随机森林的最大缺点之一，如果只想简单地拥有最佳性能的模型，并且可以牺牲解释特征的重要性，那么 PCA 可能会很有用，但并不是必须的

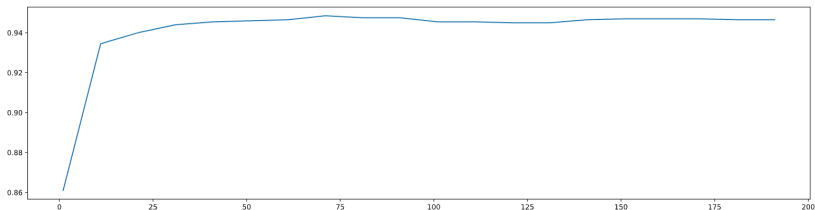
参数调优

考虑对模型在未知数据上的评估性能的影响

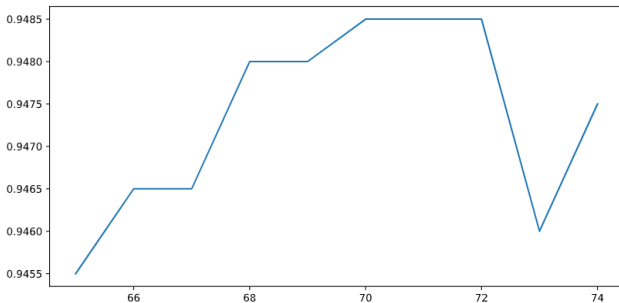
- ① 参数 `n_estimators` 的影响最大
- ② `max_depth` 次之，最大深度也代表了最高复杂度
- ③ `min_samples_leaf` 和 `min_samples_split` 并列
- ④ `max_features` 对模型的影响较小
- ⑤ `criterion` 一般使用 `gini`，具体需要视情况而定

交叉验证寻找最优 `n_estimators`

先以 10 为间隔，可以求出在 71 附近最优
准确率为 0.9484999999999999



进一步细化，在 65-75 之间以逐个进行寻找
最终得到最优的 $n_estimators$ 为 72
此时准确率提高至 0.9485000000000001



网格搜索对其余参数进行调优

内容

参考内容

- ① 随机森林 - Random Forest
- ② 决策树 VS 随机森林
- ③ 利用 AdaBoost 元算法提高分类性能
- ④ 各种机器学习算法的应用场景

Thank you!