

数据路径约定

单键

lgb决策树训练生成的东西

轻轻

lgb决策树训练生成的东西

归一化

SVM

2021年2月9日更新

2021年2月18日更新

神经网络

单按键

双按键

画图

2021年2月21日更新

double\_mixed分类

v6.1更新:

2021年2月27日更新

sensor\_classify\_default8

## 数据路径约定

- 原始数据集的名称对应关系如下
  - 单键.xlsx 对应 名称 为 single\_set.xlsx
  - 轻轻.xlsx 对应 名称 为 double\_small\_set.xlsx
- 把上述excel放在data文件夹下面
- 由于lightgbm包要求类别从0开始，所以3个力的大小对应到的是[0,1,2],而16个位置对应到的是[0,1,2...15],你不用改动，我是在读入数据的时候，把位置减1了，你还是按原来那样从1开始记录位置就行

## 单键

single是单键，命令:

先预处理:

```
python preprocess.py --category single
```

效果是这样的:

```
(venv) (base) tsq@tsq-PC: ~/PycharmProjects/05/sensor_classify$ python preprocess.py --category single
总行数:147
总列数:21
第一行:['轻\\次数', 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 'time', '', '', '']
载入数据
载入数据
Done
```

再训练（现在只实现了决策树）:

```
python train.py --category single
```

效果是这样的：

```
tsq@tsq-PC: ~/PycharmProjects/OS/sensor_classify +
(base) tsq@tsq-PC:~/PycharmProjects/OS/sensor_classify$ python train.py --category single
载入数据
数据拆分
训练集
验证集
测试集
设置参数
开始训练
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000344 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 645
[LightGBM] [Info] Number of data points in the train set: 116, number of used features: 16
[LightGBM] [Info] Start training from score -1.257083
[LightGBM] [Info] Start training from score -1.040018
[LightGBM] [Info] Start training from score -1.015921
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[1] valid_0's multi_logloss: 1.13994
Training until validation scores don't improve for 30 rounds
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[2] valid_0's multi_logloss: 1.11037
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[3] valid_0's multi_logloss: 1.11106
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[4] valid_0's multi_logloss: 1.08182
```

## lgb决策树训练生成的东西

lgb的全称是lightgbm，网上有很多它的资料，详细可以去搜一搜比如这个[博客](#)

会生成3个文件在./data/single/force/lgb下面

```
offline_test.csv
```

- 这个csv有3列：index,label,preds
  - index是验证集的一条数据在./data/single/force/raw.csv里的索引
  - label是数据的标签
  - preds是模型预测的分类结果

```
scores.txt
```

- scores.txt是调用sklearn的包测试出来的分数，采用多分类评价标准，如果最好理解的就是accuracy,即分对了有百分之多少。详细可以去搜一搜比如这个[博客](#)
- 这次的lgb在single上的对力度的预测accuracy达到了80%,而对位置的预测accuracy则更高，为0.867

```
feature_score.csv
```

- 这个csv有2列：feature,importance
  - feature是特征的名称
  - importance是lgb决策树在训练过程中学习到的特征的重要程度，importance越高，这个特征就越会被决策树重视，用来做树分支的时候就越靠近树根

## 轻轻

double\_small是轻轻，命令：

先预处理:

```
python preprocess.py --category double_small
```

效果是这样的:

```
(venv) (base) tsq@tsq-PC:~/PycharmProjects/05/sensor_classify$ python preprocess.py --category double_small
总行数:361
总列数:37
第一行:['', '', 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 'time', '', 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 0, 12.0, 13.0, 14.0, 15.0, 16.0, 'time2']
载入数据
载入数据
Done
```

再训练:

```
python train.py --category double_small
```

效果是这样的:

```
train (1) x
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[1539] valid_0's multi_logloss: 5.18681
Early stopping, best iteration is:
[1509] valid_0's multi_logloss: 5.18681
线下预测
offline.shape (36, 2)
preds_offline_df.shape (36, 1)
/home/tsq/PycharmProjects/05/venv/lib/python3.5/site-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarni
_warn_prf(average, modifier, msg_start, len(result))
/home/tsq/PycharmProjects/05/venv/lib/python3.5/site-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarni
_warn_prf(average, modifier, msg_start, len(result))
特征选择
Done
Process finished with exit code 0
```

## lgb决策树训练生成的东西

把2位置转换成了120类分类问题

- 会多生成1个文件在./data/single/force/lgb下面
  - 生成的offline\_test\_double\_position.csv里面的pos\_1\_label和pos\_2\_label是excel里的标注的2个按下去的位置

发现, 双位置训练需要一定的时间了, 大约10秒

最后的准确率非常低, 只有0.194

## 归一化

采用归一化之后, 单键的准确率保持不变, 轻轻的准确率提高到了0.25

现在默认使用归一化后的数据

如果要不用归一化, 则命令为:

```
python train.py --category double_small --not_normalize
```

## SVM

加入了svm的分类器模型，调用的命令为：

单键：

```
python train.py --category single --model svm
```

力准确率: 0.933

位置准确率: 0.533

因为double\_small标注的force都是0，而svm训练的时候要求标签不只有一类，故，只能预测position：

```
python train.py --category double_small --model svm --task position
```

哈哈，预测的准确率只有3%

## 2021年2月9日更新

按键顺序随机，按键的位置是在第A列，力度标注是在第R列

这样其实只要改动preprocess.py的creat\_dataset函数就行

现在只对单键的做了更改：

- 加入了一个category叫做single\_mixed，来表示是否用到的是mixed的乱序数据, single就还是代表上次那个每个点重复3次数据的single\_set.xlsx
- 如果需要运行single\_mixed数据集，只要如下2步：
  - 预处理：

```
python preprocess.py --category single_mixed
```

- 训练:

```
python train.py --category single_mixed --model lgb  
python train.py --category single_mixed --model svm
```

把结果与之前的比较：

category	model	task	Accuracy	测试集数据量
single_mixed	lgb	force	0.727	44
single_mixed	lgb	position	0.818	44
single_mixed	svm	force	0.750	44
single_mixed	svm	position	0.614	44
single	lgb	force	0.800	15
single	lgb	position	0.867	15

category	model	task	Accuracy	测试集数据量
single	svm	position	0.533	15

## 2021年2月18日更新

增加了双键乱序的category(DONE)

增加了神经网络模型(DONE)

## 神经网络

纠结1: batch\_size设置为多少?

ans: 先设置为1吧

纠结2: 当双键按的时候, 是否要把它归为120类分类问题? 还是有更好的办法?

或许可以argmax的时候取2个prob的位置, 算2个loss, 然后加起来再backward

试了一些比较好的参数得到的结果:

## 单按键

```
python train.py --category single_mixed --model network --lr 3e-3 --weight_decay 1e-5 --task both
```

结果会在 data/single\_mixed/force/network 和 data/single\_mixed/position/network 下面,

- 里面的每个csv文件都是形如 model\_epoch%d\_val%.3f.csv 的
  - epoch后接着的整数是表示这是用第几个epoch验证得到的结果
  - val后面接着的分数是accuracy得分
- 里面会有个log.txt文件, 就是训练的日志了, 里面标出来了训练的参数、loss等
  - 其中, 比如 progress: 0.01 loss: 0.6960 (0.8719) 就是说在0.01个epoch的时候, 从当前的样本往前数20个训练样本, 这20个里面loss的中位数大小是0.6960。而括号里的0.8719则是global\_average, 即累计所有训练样本的loss取的平均值
  - 具体可以看看utils.py是怎么写的

目前, 我试了几个学习率的参数, 你也可以调着试试, 改动命令行里的 --lr 和 --weight\_decay 就行, 单键force的预测准确率最高为 0.795, 单键position的预测准确率最高为 0.977

## 双按键

把双键处理为简单的120类分类任务:

```
python train.py --category double_small --model network --lr 5e-4 --weight_decay 3e-6 --task position --not_use_top2
```

结果会在 data/double\_small/force/network/not\_use\_top2 下面, position的预测准确率最高为 0.528

采用2个16维的prob位置向量(共32维)作为双键的预测位置的依据：

```
python train.py --category double_small --model network --lr 3e-3 --weight_decay 1e-5 --task position
```

结果会在 `data/double_small/force/network/` 下面，奇怪的是，position的预测准确率最高为0.556，并没有什么大提高

注意：

- 如果训练双键，运行的命令不含 `--not_use_top2` 的时候，会把整个 `data/double_small/force/network/` 目录删除，所以如果之前存在 `data/double_small/force/network/not_use_top2`，也会被删除。
- 最好先训练不含 `--not_use_top2` 的模型，后训练含 `--not_use_top2` 的模型。或者你把文件复制保存在别的目录下也行

## 画图

命令，只要规定好log.txt文件的位置，就可以根据此log画出loss和accuracy随着训练的变化图

例如：

```
python utils.py --log_path ./data/double_small/position/network/not_use_top2/log.txt
```

就会使用 `./data/double_small/position/network/not_use_top2/log.txt` 来画图

图片会出现在 `./figure/_data_double_small_position_network_not_use_top2_log` 下面

目前，其他几个可以画出的神经网络训练变化图如下：

```
python utils.py --log_path ./data/double_small/position/network/log.txt
python utils.py --log_path ./data/single_mixed/position/network/log.txt
python utils.py --log_path ./data/single_mixed/force/network/log.txt
```

## 2021年2月21日更新

### double\_mixed分类

流程：

- 预处理：

```
python preprocess.py --category double_mixed
```

- 训练:

```
python train.py --category double_mixed --model lgb
python train.py --category double_mixed --model svm
```

神经网络预测position:

```
python train.py --category double_mixed --model network --lr 3e-4 --
weight_decay 1e-6 --task position

python utils.py --log_path ./data/double_mixed/position/network/log.txt
```

神经网络预测force，由于只有3类，学习率太小容易过拟合:

```
python train.py --category double_mixed --model network --lr 3e-3 --
weight_decay 1e-5 --task force

python utils.py --log_path ./data/double_mixed/force/network/log.txt
```

从double\_mixed的结果看，Position准确率有一些提升空间，但是force的准确率已经很高了

## v6.1更新:

svm和lgb也采用了同样的统计方式，获得的文件名是 `val0.306_0right47_1right6_2right19.csv`

val后接的分数0.306是按  $\frac{1right_{num} + 2 \times 2right_{num}}{2 \times data_{num}}$  计算的,其道理与v5.2时说的类似

而scores.txt里的accuracy就是  $\frac{2right_{num}}{data_{num}}$

其中,  $1right_{num}$  指预测对一个键的数据数量,  $2right_{num}$  指预测对2个键的数据数量,  $data_{num}$  指数据总量

# 2021年2月27日更新

增加了新的默认 预测是8个键的包，名字叫：

## sensor\_classify\_defualt8

用这个包，运行的所有命令都与之前一样，只不过默认的预测的键盘位置是1-8，采用的电信号也是1-8的

预处理完后，force下面的raw.csv长这样，只有8列特征：

Project		raw.csv	
▼	sensor_classify_default8	1	1, 2, 3, 4, 5, 6, 7, 8, force_label, index
▼	data	2	14.6065054130575, 15.7979352881494, 35.2
▼	double_mixed	3	25.5237798791696, 16.0029859627228, 9.66
▼	force	4	32.6617062179916, 38.0084233914838, 69.7
▼	lgb	5	27.8978638689821, 39.1889361005737, 25.3
▼	network	6	33.0229116678232, 40.9433313360292, 30.2
▼	nomalized	7	18.4124045780175, 8.4405753905628, 27.02
▼	svm	8	15.1570717787954, 3.19907513663845, 3.54
	raw.csv	9	10.7589861298761, 2.69490051931922, 5.73
▼	position	10	11.2126813149979, 11.4847118721234, 23.4
▼	double_small	11	23.3327902040132, 26.9032993574814, 21.6
		12	20.5307717241983, 17.6334833263537, 18.9
		13	47.7845989696372, 11.3682179351689, 12.4
		14	15.1322575369072, 55.4539760745755, 9.56
		15	28.4549028039225, 10.5443033158184, 12.6
		16	28.6475971241863, 25.8293642846167, 27.8
		17	6.33243394965324, 47.2549724738962, 92.7