AI量化模型挑战赛

平台说明会&模型训练讲座

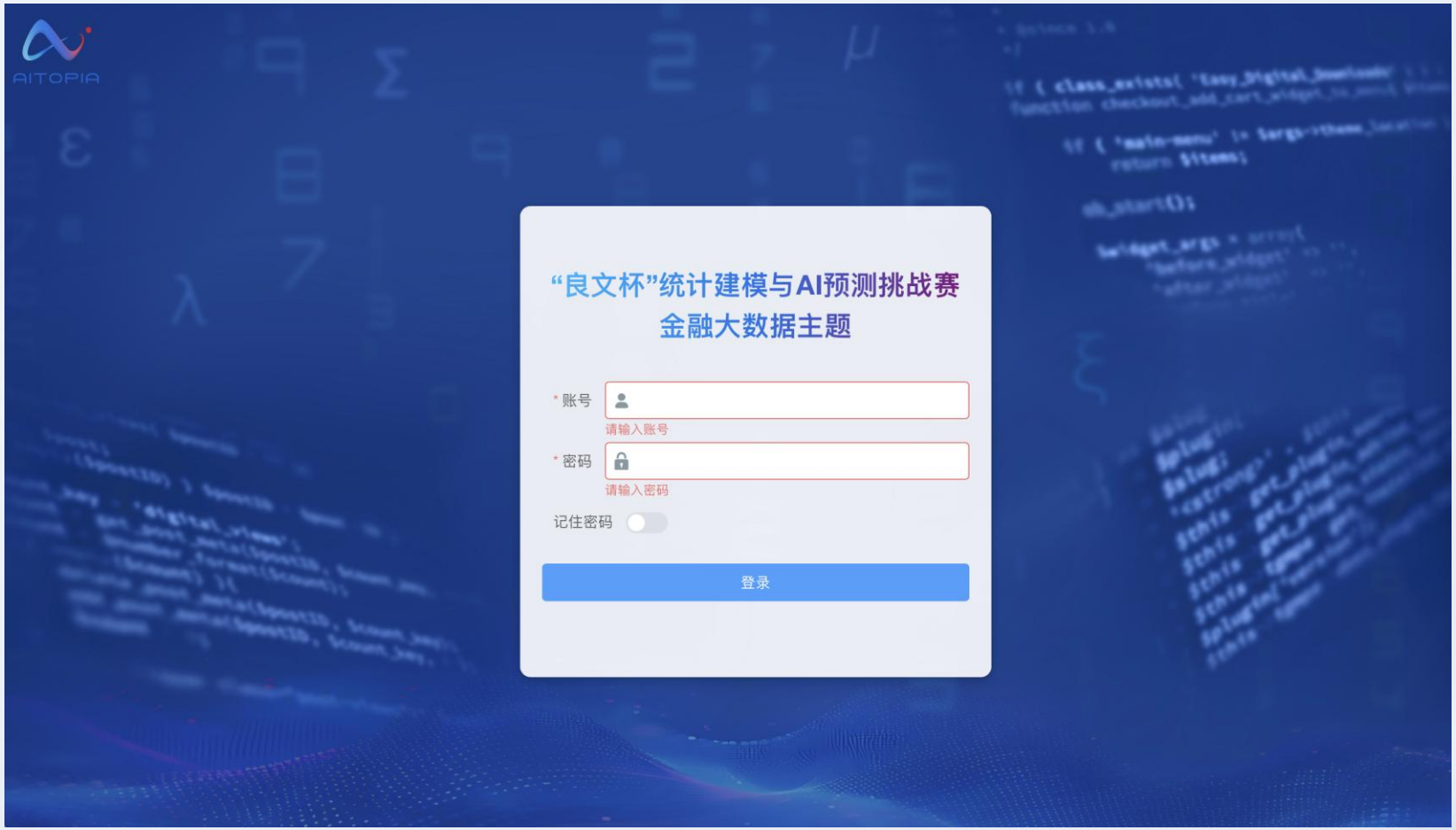
AITOPIA

目录
CONTENTS

AITOPIA

1.1 平台登录

- 平台链接： https://fin-bdc.aitopia.cn/
- 账号密码：群公告报名查询链接签收 https://web.wps.cn/etapps/query/q/jvtoUAkE



AITOPIA

1.2 平台功能

首次登录
请修改密码

指南
- 竞赛规则
- 数据介绍
- 评测指南

榜单
- 公榜评测
- 私榜评测

训练&验证：本地

测试：评测平台

根据测试结果迭代

AITOPIA

1.3 平台提交&公榜评测

- 平台提供CPU评测模式和GPU评测模式，用户可以根据模型需求选择评测模式
- 同一账号**每12小时**仅限提交1次评测任务，建议合理规划本地模型优化与平台评测节奏
  - 如果任务异常，请下载日志查看错误信息并修改bug，删除异常任务后可重新提交
  - 如提交后12小时后仍未有结果，可QQ群联系AITOPIA管理员
- 单次模型评测时长限制在**3小时**内，超时任务将自动终止并报异常
- 模型文件不超过**2GB**，FP32精度

任务状态

- 任务创建成功
- 等待调度/调度成功
- 开始创建环境/环境创建成功
- 进入评测队列/评测中/评测完成
- 发生异常，请下载日志查看错误信息

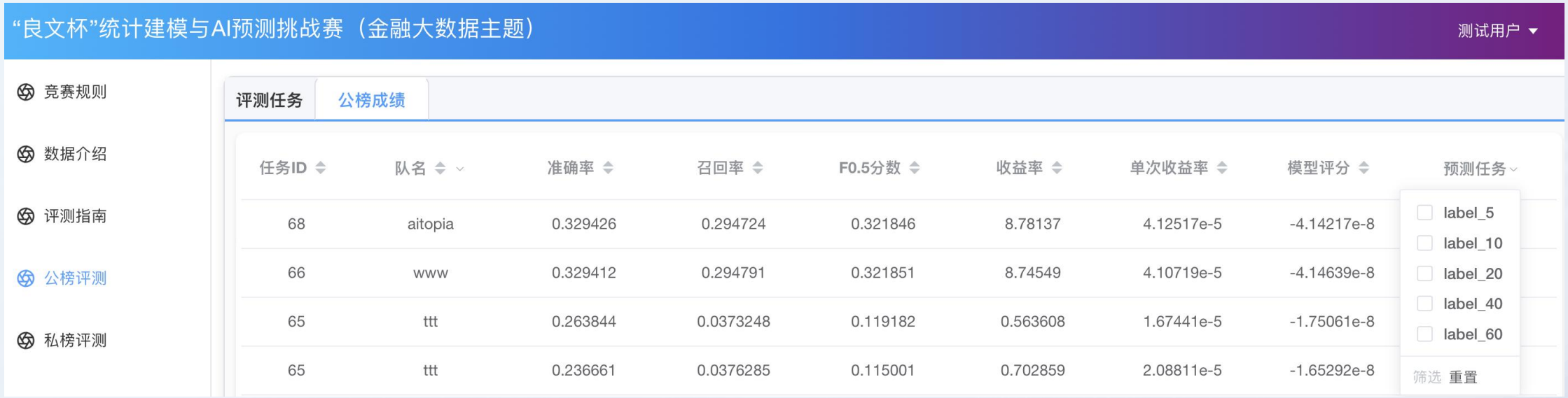
任务评测时长

- 从任务进入“评测中”到“评测完成”计时
- 不包括评测环境创建时间

AITOPIA

1.4 公榜成绩查看

- “预测任务”可筛选，其他字段可排序
- “模型评分”为公榜排行依据



1.5 综合评分标准

- 初赛模型综合评分标准：

综合评分标准：

$$\text{Final-Score} = \text{F-Score}_{\beta=0.5} \cdot (\text{pnl-average} - C)^2 \cdot sign(\text{pnl-average} - C)$$

其中，

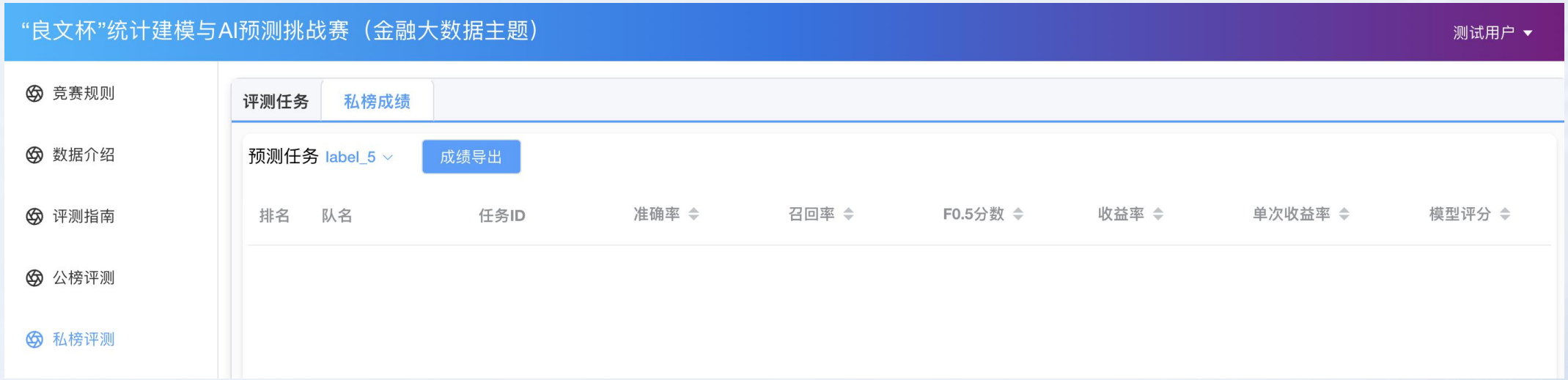
$$\text{F-Score}_{\beta} = \frac{(1+\beta\cdot\beta)\cdot \text{Precision}\cdot \text{Recall}}{\beta\cdot\beta\cdot \text{Precision} + \text{Recall}}$$

取 $\beta = 0.5$，此时更侧重准确度，兼顾召回率

  - pnl_average: 单次收益率 = 收益率除以上涨/下跌总数
  - C：所有参赛提交的平均单次收益率
  - (pnl_average - C)^2：比平均水平高出越多，奖励越大，且是平方级奖励
  - sign(): 符号函数

1.6 私榜评测

- 平台将自动选取用户在公榜评测中最后一次成功提交且结果有效的模型用于私榜评测
  - 请确保在开放提交时间段内，在公榜评测中最近创建的任务中的模型，即为参与私榜评测的最终版本
  - 可以通过删除任务，调整最终参与私榜评测的模型
- 平台调用模型时间：公榜评测结束1-2天后
- 私榜评价标准和公榜一致



AITOPIA

目录
CONTENTS

AITOPIA

2.2 提交包准备

Step0：必要文件

- 为正确评测模型，需按引导文字准备以下提交文件：
  - 1. 模型定义及权重文件:
    - model.py
    - model.pth
  - 2. 提交说明：config.json
    - python_version: 你用到了哪个python版本？
    - batch: 你每次预测多少数据量？
    - feature: 你用到了哪些数据列?
    - label: 本次提交需要测试哪个label？
  - 3. 环境依赖说明：requirements.txt
  - 4. 根据模板写自己的预测类：Predictor.py

文件夹名称可自由选择：**mmpc**
（midprice move prediction challenge）

注意

- 平台会把压缩包中文件提取到同一目录
- 为了防止python导入时路径出错，请将所有文件直接存放在同一文件夹内，不要创建子文件夹

AITOPIA

2.2 提交包准备

Step1：准备模型定义及权重

- 假如有了一个训练完的模型：
  - 将训练好的模型参数文件（例如model.pth）复制到提交文件夹（如mmpc ）目录下
  - 将模型定义抽离成文件（例如model.py），复制到提交文件夹（如mmpc ）目录下

2.2 提交包准备

Step2：通知测试平台 config.json

- 指定评测代码依赖的python版本
- 为了给同学们提供最大的自由度，平台允许在每次提交时，通过config.json决定“我要使用哪些数据，测试什么，每批次测试多少数据量”

提升数据处理效率

```json
{
    "python_version": "3.10",
    "batch": 32,
    "feature": [
        "n_bid1",
        "n_bid2",
        "n_bid3",
        "n_bsize1",
        "n_bsize2",
        "n_bsize3",
        "n_ask1",
        "n_ask2",
        "n_ask3",
        "n_asize1",
        "n_asize2",
        "n_asize3"
    ],
    "label": ["label_5"]
}
```

- 我只需要3档行情，只预测5tick后的涨跌幅
- 每批次数据量 32* pd.DataFrame(100, 12)

```json
{
    "python_version": "3.10",
    "batch": 32,
    "feature": [
        "time", "sym", "n_close",
        "n_open", "n_high", "n_low",
        "amount_delta", "n_midprice",
        "n_bid1", "n_bsize1", "n_bid2",
        "n_bsize2", "n_bid3", "n_bsize3",
        "n_bid4", "n_bsize4", "n_bid5",
        "n_bsize5", "n_ask1", "n_asize1",
        "n_ask2", "n_asize2", "n_ask3",
        "n_asize3", "n_ask4", "n_asize4",
        "n_ask5", "n_asize5"
    ],
    "label": ["label_5", "label_10", "label_20", "label_40", "label_60"]
}
```

- 我需要全部数据，同时预测所有label的涨跌幅
- 每批次数据量 32* pd.DataFrame(100, 28)

AITOPIA

2.2 提交包准备

Step3：确定有哪些依赖包

- 为了平台能够自动安装对应环境，需要同学们导出requirement.txt
  - 用于同学们在训练中，往往会用到不同版本的python package包，例如有的同学使用pytorch 1.4，有的使用pytorch 2.0.
- 建议新建推理环境，使用pipreqs包导出，命令示例如下：

```
# 创建文件夹
mkdir file_folder
# 进入文件夹
cd file_folder
# 将模型文件、Predictor.py 复制到文件夹
cp -r /path/to/model.pkl /path/to/Predictor.py .
# 生成依赖文件
pipreqs ./ --encoding=utf8 --force
# 验证依赖文件
conda create -n test python=3.10 (指定python版本)
conda activate test
pip install -r requirements.txt
```

AITOPIA

2.2 提交包准备

Step4：创建Predictor.py

根据你的模型和预处理，创建/修改Predictor.py

- 关键位置1: 导入模型定义
  - pytorch的模型保存分为保存整个模型和只保存权重
    - 建议只保存权重
  - 在Predictor.py中import模型定义
    - 采用**相对位置**，from .model import *

保存整个模型：

```
torch.save(model,'save.pt')
```

只保存训练好的权重：

```
torch.save(model.state_dict(), 'save.pt')
```

```
import os
import numpy as np
from .model import *
```

AITOPIA

2.2 提交包准备

Step4：创建Predictor.py

根据你的模型和预处理，创建/修改Predictor.py

- 关键位置2: _init_
  - 需要确保模型权重能被正常加载
  - 为了减少路径错误，采用**指定路径**

```
class Predictor():
    Tabnine | Edit | Test | Explain | Document
    def __init__(self):
        # 指定模型路径，不使用相对路径
        pth_path = os.path.join(os.path.dirname(__file__), 'model.pth')
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        # 加载模型并移动到对应设备，假设模型是整个模型保存，如果是参数字典需要初始化结构
        self.model = self.load_model(pth_path)
        self.model.to(self.device)
        self.model.eval()  # 切换到评估模式以关闭 dropout、batchnorm 等层
```

AITOPIA

2.2 提交包准备

Step4：创建Predictor.py

根据你的模型和预处理，创建/修改Predictor.py

- 关键位置3: predict()方法
  - 单次会输入形如List[pd.DataFrame]的数据：
    - List的大小为batch，即配置文件中的batch
    - pd.DataFrame中数据为过去100个tick的数据
    - pd.DataFrame的列名为配置文件中的feature
  - 函数应返回List[List[int]]格式的预测结果
    - 外层List的大小为batch，即配置文件中的batch
    - 内层List的大小为label的个数，即配置文件中的label的个数
    - int值为预测标签（标签为0/1/2）

```
Tabnine | Edit | Test | Explain | Document
def predict(self, x: List[pd.DataFrame]) -> List[List[int]]:
    # 对输入数据进行预处理
    x_hat = self.preprocess(x)
    with torch.no_grad():  # 禁用梯度计算，加速推理
        y = []
        for i in range(5):
            y_pred = self.model(x_hat).cpu().numpy()
            y.append(np.argmax(y_pred,axis=1).tolist())
        y = np.array(y).T.tolist()
    # 确保返回格式为 List[List[int]]
    if isinstance(y[0], list):
        return y
    else:
        return [y]
```

AITOPIA

2.3 验证&打包

- 如果推理环境中程序运行没有问题，就可以打包了
- 将提交文件夹（如mmpc）打包成zip文件，上传平台公榜评测

回顾：推理环境创建

```
# 创建文件夹
mkdir file_folder
# 进入文件夹
cd file_folder
# 将模型文件、Predictor.py 复制到文件夹
cp -r /path/to/model.pkl /path/to/Predictor.py .
# 生成依赖文件
pipreqs ./ --encoding=utf8 --force
# 验证依赖文件
conda create -n test python=3.10 (指定python版本)
conda activate test
pip install -r requirements.txt
```

2.4 测试方法

- 测试方式为交互式调用
  - 参赛者需要提交源代码及模型
- 代码需提供一个Predictor类，对batch组数据点进行一次预测
  - 类应至少具有_init_方法和predict方法：
    - def predict(x: List[pd.DataFrame]) -> List[List[int]]:
    - 评测程序会多次调用predict，得到预测并汇总计算P&L

确保输入输出形状的一致性

- 为了公平性与数据保密性，评测程序会打乱测试点输入顺序
  - 不同次输入的数据无先后关系
  - date：日期会置为0，无意义
  - sym：可能的范围0-4（可能有不来自于5只训练集股票的数据，仅作泛化性的额外参考，不列入最终积分）
  - time：time会保留实际交易时间，以备参赛模型会根据时段不同做推理

AITOPIA

目录
CONTENTS

AITOPIA

回顾内容

Step1：检查并分析数据

数据类型	字段	含义	说明
信息戳	date	日期	去掉实际日期，但是保留序号，取值范围0～101，保留跨标的的可比性
	time	时间戳	保留实际时间戳，3s一档行情
	sym	股票标的	仅序号，5只股票分别为0～4
成交数据	n_close	最新成交价	无量纲，以涨跌幅表示，-1～1，即跌100%到涨100%（实际取值范围更小），下同
	n_open	开盘价	
	n_high	当前最高价	
	n_low	当前最低价	
	amount_delta	成交量变化	从上个tick到当前tick发生的成交金额，单位元
五档量价数据	n_midprice	中间价	标准化后的中间价，无量纲，以涨跌幅表示
	n_bid1~n_bid5	买一价-买五价	标准化后的买一价~买五价，无量纲，以涨跌幅表示
	n_ask1~n_ask5	卖一价-卖五价	标准化后的卖一价~卖五价，无量纲，以涨跌幅表示
	n_bsize1~ n_bsize5	买一量-买五量	标准化后的买一量~买五量，无量纲，以换手率表示
	n_asize1~ n_asize5	卖一量-卖五量	标准化后的卖一量~卖五量，无量纲，以换手率表示
预测标签	label_5, label_10, label_20, label_ 40, label_60	5tick, 10tick, 20tick, 40tick, 60tick后价格移动方向	5tick,10tick,20tick,40tick,60tick后的中间价相较于当前tick中间价的移动方向，0为下跌， 1为不变，2为上涨

上节提问：n_close的实际取值范围？
-20%~20%

5个预测任务

Step1：检查并分析数据

- 是否有缺失值？ 可能挺多的，缺20%以上可以直接去掉
- 是否有异常值？ 变化幅度
- 是否有重复值？ 价格可能不变，amount不太可能
- 样本是否均衡？
- 是否需要抽样？
- 变量是否需要转换？
- 是否需要增加新的特征？

Step2：初步尝试——以SVM为例进行分类预测

- 首先导入数据
  - 一个连续交易时段只有大约2000个点（9:40-11:20，13:10-14:50）
  - 将多个交易时段拼接在一起

载入多个交易日的数据

In [2]:
```
file_dir = "./train_set/"

sym = 4
dates = list(range(12))
df = pd.DataFrame()
for date in dates:
    if (date & 1):
        file_name = f"snapshot_sym{sym}_date{date//2}_am.csv"
    else:
        file_name = f"snapshot_sym{sym}_date{date//2}_pm.csv"
    new_row = pd.read_csv(os.path.join(file_dir,file_name))
    df = pd.concat([df, new_row], ignore_index=True)
```

AITOPIA

Step2：初步尝试——以SVM为例进行分类预测

- 选择想要输入给模型的特征和标注数据
- 选择划分训练集和测试集
- 时刻注意检查数据形状！

```
In [8]:  1 train_data.flags

Out[8]:  C_CONTIGUOUS : True
         F_CONTIGUOUS : False
         OWNDATA : True
         WRITEABLE : True
         ALIGNED : True
         WRITEBACKIFCOPY : False
         UPDATEIFCOPY : False
```

```
In [5]:
1 feature_col_names = ['n_bid1','n_bid2','n_bid3','n_bid4','n_bid5',\
2                      'n_ask1','n_ask2','n_ask3','n_ask4','n_ask5']
3 label_col_name = ['label_5']
```

```
In [6]:
1 train_sample_nums = 20000
```

```
In [7]:
1 # 别忘了数据形状和存储连续性
2 train_data = np.ascontiguousarray(df[feature_col_names][:train_sample_nums].values)
3 train_label = df[label_col_name][:train_sample_nums].values.reshape(-1)
4
5 test_data = np.ascontiguousarray(df[feature_col_names][train_sample_nums:].values)
6 test_label = df[label_col_name][train_sample_nums:].values.reshape(-1)
```

AITOPIA

Step2：初步尝试——以SVM为例进行分类预测

- 做数据分析一定要了解数据！

别忘了看一下标签分布：

In [9]:

```
1 print("在训练集中：")
2 print("标签为0的样本个数：", sum(train_label == 0))
3 print("标签为1的样本个数：", sum(train_label == 1))
4 print("标签为2的样本个数：", sum(train_label == 2))
5
6 print("在测试集中：")
7 print("标签为0的样本个数：", sum(test_label == 0))
8 print("标签为1的样本个数：", sum(test_label == 1))
9 print("标签为2的样本个数：", sum(test_label == 2))
```

```
在训练集中：
标签为0的样本个数： 4274
标签为1的样本个数： 11493
标签为2的样本个数： 4233
在测试集中：
标签为0的样本个数： 652
标签为1的样本个数： 2689
标签为2的样本个数： 647
```

显著不均匀

AITOPIA

Step2：初步尝试——以SVM为例进行分类预测

- “人生苦短，我用python”

导入sklearn包：

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

建立svc模型并fit：

```
In [10]:
# 训练一个native svm分类器:
model = SVC()
model.fit(train_data,train_label)

Out[10]: SVC()
```

用训练好的模型做预测：

```
In [11]:
# 训练完了之后,看看在训练集的准确率（拟合的怎么样）
y_hat = model.predict(train_data)
y = train_label
```

AITOPIA

Step2：初步尝试——以SVM为例进行分类预测

- 模型评价指标：如何评价模型结果好坏

$$\text{precision} = \frac{true\ positive}{true\ positive + false\ positive} = \frac{true\ positive}{no.of\ predicted\ positive}$$

$$\text{recall} = \frac{true\ positive}{true\ positive + false\ negative} = \frac{true\ positive}{no.of\ actual\ positive}$$

$$\text{F1Score} = 2 \times \frac{precision \times recall}{precision + recall}$$

交易侧重precision 说的最好是真的,用F0.5

Step2：初步尝试——以SVM为例进行分类预测

- 模型评价指标：如何评价模型结果好坏
  - 本次比赛采用F0.5作为评价指标的一部分
  - 考虑上涨/下跌情况的准确率和召回率，并计算F0.5系数

$$F_\beta = (1+\beta^2) \times \frac{precision \times recall}{\beta^2 \times precision + recall}$$

回顾：综合评分标准

综合评分标准：

$$\text{Final-Score} = \text{F-Score}_{\beta=0.5} \cdot (\text{pnl-average} - C)^2 \cdot sign(\text{pnl-average} - C)$$

其中，

$$\text{F-Score}_\beta = \frac{(1+\beta \cdot \beta) \cdot \text{Precision} \cdot \text{Recall}}{\beta \cdot \beta \cdot \text{Precision} + \text{Recall}}$$

取 $\beta = 0.5$，此时更侧重准确度，兼顾召回率

AITOPIA

Step2：初步尝试——以SVM为例进行分类预测

- 结果不好怎么办？
  - 分析原因
    - 模型导致的？
    - 参数导致的？
  - 修改模型
  - 参数调整/搜索

In [12]:

```
# 总体情况
print("预测正确的标签数：", sum(y_hat == y))
print("总体正确率：", sum(y_hat == y)/len(y_hat))

# 分标签查看：
print("真实标签为0样本的正确预测个数：", sum(y[y == 0] == y_hat[y == 0]))
print("真实标签为1样本的正确预测个数：", sum(y[y == 1] == y_hat[y == 1]))
print("真实标签为2样本的正确预测个数：", sum(y[y == 2] == y_hat[y == 2]))

## 我们更关心上涨下跌情况的预测
# 所有不为1的标签的召回率（即仅考虑真实标签为上涨或下跌样本是否被正确分类）
index = y != 1
print("上涨下跌召回率：", sum(y_hat[index]==y[index])/sum(index))
# 所有不为1的标签的准确率（即仅考虑预测为上涨或下跌样本是否是正确）
index = y_hat != 1
print("上涨下跌准确率：", sum(y_hat[index]==y[index])/sum(index))
```

```
预测正确的标签数： 11493
总体正确率： 0.57465
真实标签为0样本的正确预测个数： 0
真实标签为1样本的正确预测个数： 11493
真实标签为2样本的正确预测个数： 0
上涨下跌召回率： 0.0
上涨下跌准确率： nan
```

别忘了看一下标签分布：

In [9]:

```
print("在训练集中：")
print("标签为0的样本个数：", sum(train_label == 0))
print("标签为1的样本个数：", sum(train_label == 1))
print("标签为2的样本个数：", sum(train_label == 2))

print("在测试集中：")
print("标签为0的样本个数：", sum(test_label == 0))
print("标签为1的样本个数：", sum(test_label == 1))
print("标签为2的样本个数：", sum(test_label == 2))
```

```
在训练集中：
标签为0的样本个数： 4274
标签为1的样本个数： 11493
标签为2的样本个数： 4233
在测试集中：
标签为0的样本个数： 652
标签为1的样本个数： 2689
标签为2的样本个数： 647
```

AITOPIA

Step3：模型优化

- 可能的问题1：类别不均匀，静止tick太多
- 可能的解决办法：
  - 根据样本频率做加权

In [15]:
```
## 对样本依据类别加权：
model2 = SVC(class_weight='balanced')
model2.fit(train_data,train_label)
```

Out[15]:
```
SVC(class_weight='balanced')
```

- 结果：总体准确率下降了，但是只考虑上涨/下跌的情况，召回率和准确率都有大幅提升

In [17]:
```
# 总体准确率：
print("总体准确率：", sum(y_hat == y)/len(y_hat))
# 所有不为1的标签的召回率（即仅考虑真实标签为上涨或下跌样本是否被正确分类）
index = y != 1
print("训练集上涨下跌召回率：", sum(y_hat[index]==y[index])/sum(index))
# 所有不为1的标签的准确率（即仅考虑预测为上涨或下跌样本是否是正确）
index = y_hat != 1
print("训练集上涨下跌准确率：", sum(y_hat[index]==y[index])/sum(index))
```

```
总体准确率：  0.4567
训练集上涨下跌召回率：  0.28329610908663455
训练集上涨下跌准确率：  0.25706666666666667
```

AITOPIA

Step3：模型优化

- 可能的问题1：类别不均匀，静止tick太多

- 可能的问题2：样本稀疏
- 可能的问题3：存在噪音（误分类点与惩罚系数）
- 可能的问题4：……

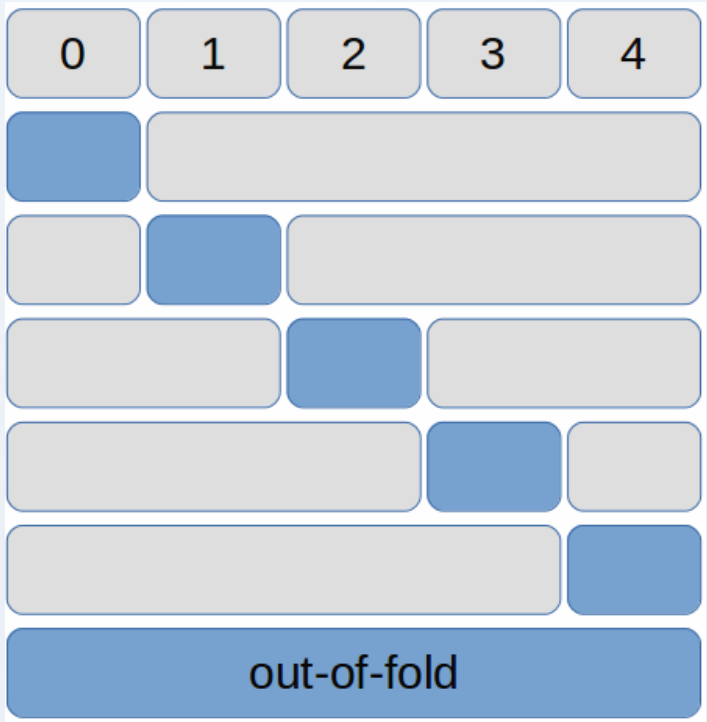
Step3：模型优化

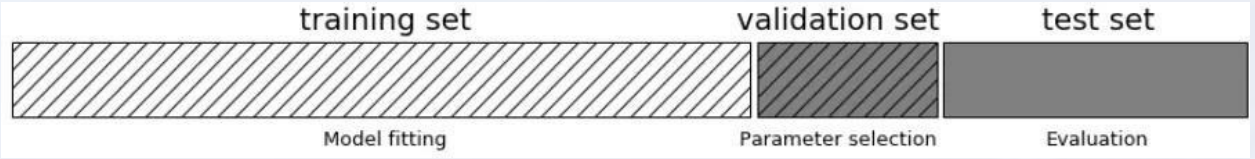
- 如何进行参数的排列组合搜索？
  - GridSearch：网格搜索

	C = 0.001	C = 0.01	…	C = 10
gamma=0.001	SVC(C=0.001, gamma=0.001)	SVC(C=0.01, gamma=0.001)	…	SVC(C=10, gamma=0.001)
gamma=0.01	SVC(C=0.001, gamma=0.01)	SVC(C=0.01, gamma=0.01)	…	SVC(C=10, gamma=0.01)
…	…	…	…	…
gamma=100	SVC(C=0.001, gamma=100)	SVC(C=0.01, gamma=100)	…	SVC(C=10, gamma=100)

Step3：模型优化

- 如何进行参数的排列组合搜索？
  - GridSearch：网格搜索
  - CrossValidation：交叉验证





Step3：模型优化

- 如何进行参数的排列组合搜索？
  - GridSearch：网格搜索
  - CrossValidation：交叉验证
  - sklearn.model_selection. GridSearchCV

问题2： SVM参数这么多，有没有什么简单的参数调整方法?

In [20]:
```
model = SVC()
grid_params = [{'kernel':['rbf','linear'],'C':[0.5,1,5], 'class_weight':['balanced']}]
Grid = GridSearchCV(model, grid_params, cv = 5, scoring = 'accuracy',refit=True, n_jobs = 6)
Grid.fit(train_data,train_label)
```

Out[20]:
```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=6,
             param_grid=[{'C': [0.5, 1, 5], 'class_weight': ['balanced'],
                          'kernel': ['rbf', 'linear']}],
             scoring='accuracy')
```

AITOPIA

Step 4：特征工程（Featuer Engeering）

- 如何把先验信息加入模型中？
  - 特征(自变量)
- 增加有效信息：
  - 根据领域知识加入更有效的特征/特征组合
- 去除无效干扰：
  - 去除数据噪音：
  - 降维：PCA
  - Normalization：Z-score

Step 4：特征工程（Featuer Engeering）

- 做一些简单处理

提供的数据已经做了处理 不再是价格而是相对前收盘的净值,量也取过对数了

```
# 价格+1（从涨跌幅还原到对前收盘价的比例）
new_df['bid1'] = new_df['n_bid1']+1
new_df['bid2'] = new_df['n_bid2']+1
new_df['bid3'] = new_df['n_bid3']+1
new_df['bid4'] = new_df['n_bid4']+1
new_df['bid5'] = new_df['n_bid5']+1
new_df['ask1'] = new_df['n_ask1']+1
new_df['ask2'] = new_df['n_ask2']+1
new_df['ask3'] = new_df['n_ask3']+1
new_df['ask4'] = new_df['n_ask4']+1
new_df['ask5'] = new_df['n_ask5']+1
```

```
# 对量取对数
new_df['bsize1'] = new_df['n_bsize1'].map(np.log)
new_df['bsize2'] = new_df['n_bsize2'].map(np.log)
new_df['bsize3'] = new_df['n_bsize3'].map(np.log)
new_df['bsize4'] = new_df['n_bsize4'].map(np.log)
new_df['bsize5'] = new_df['n_bsize5'].map(np.log)
new_df['asize1'] = new_df['n_asize1'].map(np.log)
new_df['asize2'] = new_df['n_asize2'].map(np.log)
new_df['asize3'] = new_df['n_asize3'].map(np.log)
new_df['asize4'] = new_df['n_asize4'].map(np.log)
new_df['asize5'] = new_df['n_asize5'].map(np.log)
new_df['amount'] = new_df['amount_delta'].map(np.log1p)
```

```
# 量价组合
new_df['spread1'] =  new_df['ask1'] - new_df['bid1']
new_df['spread2'] =  new_df['ask2'] - new_df['bid2']
new_df['spread3'] =  new_df['ask3'] - new_df['bid3']
new_df['mid_price1'] =  new_df['ask1'] + new_df['bid1']
new_df['mid_price2'] =  new_df['ask2'] + new_df['bid2']
new_df['mid_price3'] =  new_df['ask3'] + new_df['bid3']
new_df['weighted_ab1'] = (new_df['ask1'] * new_df['n_bsize1'] + new_df['bid1'] * new_df['n_asize1']) / (new_df
new_df['weighted_ab2'] = (new_df['ask2'] * new_df['n_bsize2'] + new_df['bid2'] * new_df['n_asize2']) / (new_df
new_df['weighted_ab3'] = (new_df['ask3'] * new_df['n_bsize3'] + new_df['bid3'] * new_df['n_asize3']) / (new_df

new_df['relative_spread1'] = new_df['spread1'] / new_df['mid_price1']
new_df['relative_spread2'] = new_df['spread2'] / new_df['mid_price2']
new_df['relative_spread3'] = new_df['spread3'] / new_df['mid_price3']
```

```
# 均线特征
new_df['ask1_ma5']  = new_df['ask1'].rolling(window=5,  min_periods=1).mean()
new_df['ask1_ma10'] = new_df['ask1'].rolling(window=10, min_periods=1).mean()
new_df['ask1_ma20'] = new_df['ask1'].rolling(window=20, min_periods=1).mean()
new_df['ask1_ma40'] = new_df['ask1'].rolling(window=40, min_periods=1).mean()
new_df['ask1_ma60'] = new_df['ask1'].rolling(window=60, min_periods=1).mean()
new_df['bid1_ma5']  = new_df['bid1'].rolling(window=5,  min_periods=1).mean()
new_df['bid1_ma10'] = new_df['bid1'].rolling(window=10, min_periods=1).mean()
new_df['bid1_ma20'] = new_df['bid1'].rolling(window=20, min_periods=1).mean()
new_df['bid1_ma40'] = new_df['bid1'].rolling(window=40, min_periods=1).mean()
new_df['bid1_ma60'] = new_df['bid1'].rolling(window=60, min_periods=1).mean()
```

为简单模型提供更多feature,复杂的模型可能自己就能抽取出来这些

AITOPIA

Step 4：特征工程（Featuer Engeering）

- 再次训练，效果大幅提升

In [10]:

```
%%time
## 对样本依据类别加权：
model = SVC(class_weight='balanced')
model.fit(train_data,train_label)
```

```
CPU times: user 2min 10s, sys: 395 ms, total: 2min 10s
Wall time: 2min 10s
```

Out[10]:

```
SVC(class_weight='balanced')
```

In [11]:

```
# 训练集
y_hat = model.predict(train_data)
y = train_label
# 总体准确率：
print("总体准确率：", sum(y_hat == y)/len(y_hat))
# 所有不为1的标签的召回率（即仅考虑真实标签为上涨或下跌样本是否被正确分类）
index = y != 1
print("训练集上涨下跌召回率：", sum(y_hat[index]==y[index])/sum(index))
# 所有不为1的标签的准确率（即仅考虑预测为上涨或下跌样本是否是正确）
index = y_hat != 1
print("训练集上涨下跌准确率：", sum(y_hat[index]==y[index])/sum(index))
```

```
总体准确率： 0.5074
训练集上涨下跌召回率： 0.568785523721146
训练集上涨下跌准确率： 0.3862544308782985
```

AITOPIA

简单回顾

- 我们做了什么操作？
  - 观察数据
  - 清洗和整理输入数据（/标注）
  - 选择一个合适的模型进行训练
  - 根据训练结果和数据本身的特性进行参数优化

简单回顾

- 我们还可以做什么？
  - 更换其他的模型：决策树，随机森林，Adaboost等等

```
%pylab inline
import pandas as pd
import numpy as np
import time
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import (RandomForestClassifier, ExtraTreesClassifier, AdaBoostClassifier,\
                              GradientBoostingClassifier)
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
```

简单回顾

- 我们还可以做什么？
  - 更换其他的模型：决策树，随机森林，Adaboost等等
  - 模型集成：
    - 用多个模型进行投票，选择最终结果
    - Boosting/Bagging
  - 更精细的处理输入数据
    - 合适的标准化：min-max；z-score；
    - 维度增加/减少，分析不同特征的重要性

THANKS !