



开始 批注 编辑 格式转换 阅读 效率工具 会员专享

← → 滚屏 PDF转Word PDF转图片

批注 编辑 调整页面 提取页面 拆分合并 提取文字 提取表格 提取图片 全文翻译 压缩文档 快捷工具 插件 打印

缩略图 目录



1



2



3



4

Kaggle 内部材料 禁止复制及售卖

kaggle 金融量化比赛

Jane Street 简街市场预测竞赛详解

比赛名称: Jane Street Market Prediction**比赛链接:** <https://www.kaggle.com/c/jane-street-market-prediction/>**比赛类型:** 金融时间序列、匿名结构化

比赛背景

在交易市场上买卖双方将做出理性交易决策，制定交易策略以识别和利用低效率是具有挑战性的，选手将使证券交易所的市场数据构建自己的量化交易模型，以最大限度地提高回报。

比赛任务

您面临的挑战是使用您可以使用的历历史数据、数学和模型来创建一个尽可能接近确定性的模型。您将看到许多潜在的交易机会，您的模型必须选择接受或拒绝这些机会。

评价指标

测试集中的每一行代表一个交易机会，您将为其预测一个 **action** 值，1 表示进行交易，0 表示不进行交易。

每笔交易都有一个关联的 **weight** 和 **resp**，代表权重和回报信息。

$$p_i = \sum_j (weight_{ij} * resp_{ij} * action_{ij})$$

竞赛数据

训练数据 竞赛数据集包含一组脱敏特征，代表真实的股票市场数据。数据集中的每一行代表一个交易机会，将为此预测一个 action 值：1 表示进行交易，0 表示跳过。每笔交易都有一个关联的 weight 和 resp，它们共同代表交易的回报。date 列是一个整数，代表交易日期，而 ts_id 列代表时间排序。除了匿名特征值之外，还可以获得有 features.csv 中特征的元数据。在训练集 train.csv 中，会提供了一个 resp 值，以及 resp_{1,2,3,4} 代表不同时间范围内回报的其他几个值。这些变量不包含在测试集中。为完整性起见，交易 weight=0 故意包含在数据集中，尽管此类交易不会对评分评估做出贡献。

测试数据 在比赛的模型训练阶段，这个看不见的测试集由大约 100 万行历史数据组成。在实时预测阶段，测试集将使用定期更新的实时市场数据。

比赛总结

简街比赛是一场匿名时序结构化比赛，需要选手按照时间序列进行预测，比赛同时也是金融量化类型比赛。

对于训练集和测试集每一条记录，需要根据 feature_{0...129} 来预测具体的 action 行为。

简街比赛的难点如下：

- 训练集并没有直接给定 action 标签，需要从 resp_{1,2,3,4} 构造；
- 数据集特征都是匿名的，虽然给定了元信息，但是很难做有效的特征工程；
- 数据集按照时间进行组织，交叉验证需要按照时间进行划分；

在解决简街比赛中，绝大部分的选手都是使用神经网络，由于赛题本身是匿名数据，所以调参就变成了本次比赛的主旋律。

第 1 名思路

<https://zhuanlan.zhihu.com/p/355606168>

<https://www.kaggle.com/c/jane-street-market-prediction/discussion/224348>

在参赛过程中发现，如果过于调参或者只选择几折模型进行预测，在公开榜单的效果会很好。因为只用到了少量的数据，很可能产生过拟合公榜的情况。

此外评分公式考察了模型收益的稳定性，即模型的最大回撤不能过大。官方在数据中提供了 5 组收益指标，分别是 resp_1, resp_2, resp_3, resp_4 和 resp。我们的最终收益是按照 resp 的值计算的。

- 根据 resp_1, resp_2 交易所取得的收益更加稳定，t 值高
- 根据 resp_3 交易所可能有较高的 t 值，p 值也往往高
- 根据 resp, resp_4 交易可能取得很高的 p 值，但是 t 值往往很低

从以上结论我们发现了，利用 resp_3，我们可以得到既稳定，又有高收益的模型，也就是说==如果我们在预测和训练的时候直接使用 resp_3，很可能得到比用 resp 预测更高的收益！

我们最终模型是 AE+MLP + XGBOOST(100 round)，每种模型各针对三个种子进行了训练。取得的公榜分数分别是 99xx 和 97xx，私榜分数是 54xx 和 52xx。

第 3 名

<https://www.kaggle.com/c/jane-street-market-prediction/discussion/224713>

数据操作：

- Remove first 85 days
- Rows with weight=0 were included
- NaN filling: median conditioned on f0

模型操作：

- input block: batchnorm + logarithmic feature extension
- block 0: 3 dense layers with 100 units, batchnorm, dropout (0.35) and mish activation
- block 1-23: 2 dense layers with 100 units, batchnorm, dropout (0.35) and mish activation
- skip connections between Block 0 and Block i
- output: Dense layer with 5 units
- 15 ensembles of these deep models

最终模型预测使用 `tf-lite` 进行加速。

第 10 名

- Geometric Brownian Motion with Drift Fitting

金融资产建模时使用的最基本模型之一是具有漂移的几何布朗运动。假设资产遵循几何布朗运动分布以及漂移参数和 resp 大于 0 的概率，这给了我 resp 的预期值，所有这些都可以用来训练。

- Mixture Density Network with negative log likelihood loss function

第 19 名

<https://www.kaggle.com/zhaonat/multilabel-fold-ensemble>

使用 LightGBM 对 resp_1', 'resp_2', 'resp_3', 'resp' 进行分开进行训练：

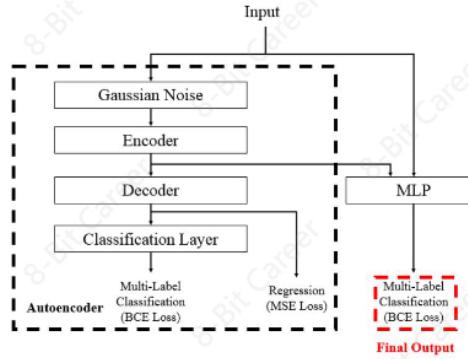
```
train['resp'] = (((train['resp'].values)*train['weight']) > 0).astype(int)
train['resp_1'] = (((train['resp_1'].values)*train['weight']) > 0).astype(int)
train['resp_2'] = (((train['resp_2'].values)*train['weight']) > 0).astype(int)
train['resp_3'] = (((train['resp_3'].values)*train['weight']) > 0).astype(int)
train['resp_4'] = (((train['resp_4'].values)*train['weight']) > 0).astype(int)
```

数据划分方法使用 GroupKFOLD，在预测阶段使用 treelite 进行加速。

其他选手分享

- 第 24 名: <https://www.kaggle.com/wendefariaslopes/nn-branch-model>
- 第 40 名: <https://www.kaggle.com/c/jane-street-market-prediction/discussion/224029>
- 第 44 名: <https://www.kaggle.com/c/jane-street-market-prediction/discussion/227167>

模型整体采用了 AutoEncoder 以及 MLP 的架构，整体框架如下图所示：



其中，Autoencoder 部分采用了去噪自编码器（Denoise Autoencoder），即在输入数据进入 Encoder 之前，先加入高斯噪声，这样起到数据增强的作用，有助于缓解过拟合，同时也可以提高特征的表示能力。自编码器的 Loss 包括两个部分，一部分是 MSE Loss，用于重构数据；另一部分是经过一个分类器模块进行多分类任务，用来实现 action 的分类，保证 Encoder 学习得到的特征具有跟交易动作相关的特性。MLP 部分的输入包括两个部分，即原始数据输入与 Encoder 的编码结果，多层 MLP 得到的结果用于最终交易动作的分类。

这部分的实现代码如下，完整源码可以去该 notebook 下获取：

<https://www.kaggle.com/code/gogo827jz/jane-street-supervised-autoencoder-mlp/notebook?scriptVersionId=73762661>

```
1 def create_ae_mlp(num_columns, num_labels, hidden_units, dropout_rates, ls =  
2  
3     inp = tf.keras.layers.Input(shape = (num_columns, ))  
4     x0 = tf.keras.layers.BatchNormalization()(inp)  
5  
6     encoder = tf.keras.layers.GaussianNoise(dropout_rates[0])(x0)  
7     encoder = tf.keras.layers.Dense(hidden_units[0])(encoder)  
8     encoder = tf.keras.layers.BatchNormalization()(encoder)  
9     encoder = tf.keras.layers.Activation('swish')(encoder)  
10  
11     decoder = tf.keras.layers.Dropout(dropout_rates[1])(encoder)  
12     decoder = tf.keras.layers.Dense(num_columns, name = 'decoder')(decoder)  
13  
14     x_ae = tf.keras.layers.Dense(hidden_units[1])(decoder)  
15     x_ae = tf.keras.layers.BatchNormalization()(x_ae)  
16     x_ae = tf.keras.layers.Activation('swish')(x_ae)  
17     x_ae = tf.keras.layers.Dropout(dropout_rates[2])(x_ae)  
18  
19     out_ae = tf.keras.layers.Dense(num_labels, activation = 'sigmoid', name =  
20  
21     x = tf.keras.layers.concatenate([x0, encoder])  
22     x = tf.keras.layers.BatchNormalization()(x)  
23     x = tf.keras.layers.Dropout(dropout_rates[3])(x)
```

```
25     for i in range(2, len(hidden_units)):
26         x = tf.keras.layers.Dense(hidden_units[i])(x)
27         x = tf.keras.layers.BatchNormalization()(x)
28         x = tf.keras.layers.Activation('swish')(x)
29         x = tf.keras.layers.Dropout(dropout_rates[i + 2])(x)
30
31     out = tf.keras.layers.Dense(num_labels, activation = 'sigmoid', name = 'ae_out')
32
33     model = tf.keras.models.Model(inputs = inp, outputs = [decoder, out_ae, out])
34     model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = lr),
35                   loss = {'decoder': tf.keras.losses.MeanSquaredError(),
36                            'ae_action': tf.keras.losses.BinaryCrossentropy(label_smoothing = 0.1),
37                            'action': tf.keras.losses.BinaryCrossentropy(label_smoothing = 0.1)},
38                   metrics = {'decoder': tf.keras.metrics.MeanAbsoluteError(name = 'MAE'),
39                             'ae_action': tf.keras.metrics.AUC(name = 'AUC'),
40                             'action': tf.keras.metrics.AUC(name = 'AUC'),
41                           })
42
43
44     return model
```

模型训练的时候采用了 PurgedGroupTimeSeriesSplit 方式对训练数据进行划分，它可以保证不会有未来数据的泄露，同时也可采用 n-folds 的方式进行交叉验证，实验时作者采用了 5-folds 的方式。

```

1 if not TEST:
2     scores = []
3     batch_size = 4096
4     gkf = PurgedGroupTimeSeriesSplit(n_splits = n_splits, group_gap = group_g
5     for fold, (tr, te) in enumerate(gkf.split(train['action'].values, train[
6         ckp_path = f'JSMModel_{fold}.hdf5'
7         model = create_ae_mlp(**params)
8         ckp = Modelcheckpoint(ckpt_path, monitor = 'val_action_AUC', verbose =
9             save_best_only = True, save_weights_only = True
10            es = EarlyStopping(monitor = 'val_action_AUC', min_delta = 1e-4, pat
11                baseline = None, restore_best_weights = True, verb
12                history = model.fit(X[tr], [X[tr], y[tr]], validation_data = (
13                    sample_weight = sw[tr],
14                    epochs = 100, batch_size = batch_size, callbacks
15                    hist = pd.DataFrame(history.history)
16                    score = hist['val_action_AUC'].max()
17                    print(f'Fold {fold} ROC AUC:\t', score)
18                    scores.append(score)
19
20        K.clear_session()
21        del model
22        rubbish = gc.collect()
23
24    print('Weighted Average CV Score:', weighted_average(scores))

```

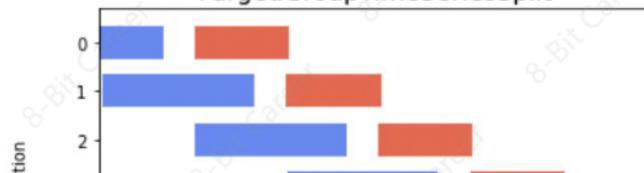
除此之外，作者还采用了一些其他 tricks，如 Early-Stop、BatchNormalization 以及超参数搜索等，另外，自编码器的激活函数用到了 swish，而不是 relu 或者 leaky-relu，不过这里具作者所言，采用哪个激活函数区别不大。

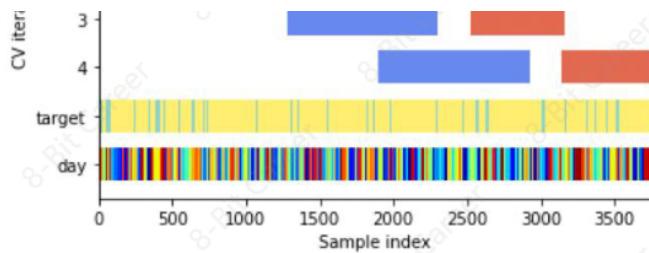
有用的技巧

- 交叉验证，由于金融任务存在任务数据是时间序列，模型的训练和使用存在时间延迟。作者使用了 PurgedGroupTimeSeriesSplit 进行交叉验证划分。

如下图所示，PurgedGroupTimeSeriesSplit 支持对时序数据进行划分，避免了验证集使用在训练集之前的数据，同时支持定义训练集和验证集之间的时间间隔。本文使用了 5-折 31-间隔的交叉验证划分方式。

PurgedGroupTimeSeriesSplit





```
import numpy as np
from sklearn.model_selection import KFold
from sklearn.model_selection._split import _BaseKFold, indexable, _num_samples
from sklearn.utils.validation import _deprecate_positional_args

# modified code for group gaps; source
# https://github.com/getgaurav2/sklearn-learn/blob/d4a3ef5cc9de3a76f0266932644b884c99724c57/sklearn/_model_selection/_validation.py
class PuredGroupTimeSeriesSplit(_BaseKFold):
    """
    Parameters
    -----
    n_splits : int, default=5
        Number of splits. Must be at least 2.
    max_train_group_size : int, default=Inf
        Maximum group size for a single training set.
    group_gap : int, default=None
        Gap between train and test
    max_test_group_size : int, default=Inf
        We discard this number of groups from the end of each train split
    """
```

```
@_deprecate_positional_args
def __init__(self,
             n_splits=5,
             *,
             max_train_group_size=np.inf,
             max_test_group_size=np.inf,
             group_gap=None,
             verbose=False
            ):
    super().__init__(n_splits, shuffle=False, random_state=None)
    self.max_train_group_size = max_train_group_size
    self.group_gap = group_gap
    self.max_test_group_size = max_test_group_size
    self.verbose = verbose

    def split(self, X, y=None, groups=None):
        """Generate indices to split data into training and test set.

        Parameters
        -----
        X : array-like of shape (n_samples, n_features)
            Training data, where n_samples is the number of samples
            and n_features is the number of features.
        y : array-like of shape (n_samples,)
            Always ignored, exists for compatibility.
        groups : array-like of shape (n_samples,)
            Group labels for the samples used while splitting the dataset into
            train/test set.

        Yields
        -----
        train : ndarray
            The training set indices for that split.
        test : ndarray
            The testing set indices for that split.

```

```
if groups is None:
    raise ValueError(
        "The 'groups' parameter should not be None")
X, y, groups = indexable(X, y, groups)
n_samples = _num_samples(X)
n_splits = self.n_splits
group_gap = self.group_gap
max_test_group_size = self.max_test_group_size
max_train_group_size = self.max_train_group_size
n_folds = n_splits + 1
group_dict = {}
```

```
        u, ind = np.unique(groups, return_index=True)
        unique_groups = u[np.argsort(ind)]
        n_samples = _num_samples(X)
        n_groups = _num_samples(unique_groups)
        for idx in np.arange(n_samples):
            if (groups[idx] in group_dict):
                group_dict[groups[idx]].append(idx)
            else:
                group_dict[groups[idx]] = [idx]
        if n_folds > n_groups:
            raise ValueError(
                ("Cannot have number of folds={0} greater than"
                 " the number of groups={1}").format(n_folds,
                                                     n_groups))
```

```
group_test_size = min(n_groups // n_folds, max_test_group_size)
group_test_starts = range(n_groups - n_splits * group_test_size,
                         n_groups, group_test_size)
for group_test_start in group_test_starts:
    train_array = []
    test_array = []

    group_st = max(0, group_test_start - group_gap - max_train_group_size)
    for train_group_idx in unique_groups[group_st:(group_test_start - group_gap)]:
        train_array_tmp = group_dict[train_group_idx]

        train_array = np.concatenate((train_array,
                                     train_array_tmp),
                                    axis=None), axis=None)

    train_end = train_array.size

    for test_group_idx in unique_groups[group_test_start:
                                         group_test_start +
                                         group_test_size]:
        test_array_tmp = group_dict[test_group_idx]
        test_array = np.concatenate((test_array,
                                    test_array_tmp),
                                   axis=None), axis=None)

    test_array = test_array[group_gap:]

    if self.verbose > 0:
        pass
```

2. 使用 swish 激活函数而不是 ReLU 来防止“死神经元”并平滑梯度；
3. 用 3 个不同的随机种子训练模型并取平均值以减少预测方差；
4. 仅使用在最后两次 交叉验证拆分中训练的模型（具有不同种子），因为它们使用了较多的数据；
5. 通过 MLP 的 BCE 损失进行早停；
6. 使用 Hyperopt 找到最优的超参数集。



Xgboost 方案

Xgboost 比较简单，指定超参，使用不同的随机数种子训练了三个模型进行使用。代码如下：

```
X = train.loc[:, train.columns.str.contains('feature')].values
y = train.loc[:, 'action'].astype('int').values

X_ = X
y_ = train.loc[:, 'action3'].astype('int').values

clf1 = xgb.XGBClassifier(
    n_estimators=100,
    max_depth=11,
    learning_rate=0.05,
    subsample=0.90,
    colsample_bytree=0.7,
    missing=-999,
    random_state=21,
    tree_method='gpu_hist', # THE MAGICAL PARAMETER
    reg_alpha=10,
    reg_lambda=10,
)
clf1.fit(X_, y_)

clf2 = xgb.XGBClassifier(
    n_estimators=100,
    max_depth=11,
    learning_rate=0.05,
    subsample=0.90,
    colsample_bytree=0.7,
    missing=-999,
    random_state=210,
    tree_method='gpu_hist', # THE MAGICAL PARAMETER
    reg_alpha=10,
    reg_lambda=10,
)
clf2.fit(X_, y_)
```

```
clf3 = xgb.XGBClassifier(
    n_estimators=100
```



```
    n_estimators=100,
    max_depth=11,
    learning_rate=0.05,
    subsample=0.90,
    colsample_bytree=0.7,
    missing=-999,
    random_state=2010,
    tree_method='gpu_hist', # THE MAGICAL PARAMETER
    reg_alpha=10,
    reg_lambda=10,
)
clf3.fit(X_, y_)
```

Kaggle TOP1：神奇的时序 Trick 直接炸榜

作者介绍

王明杰，北京师范大学珠海校区研究助理。2021 年搜狐校园文本匹配算法大赛第二名获得者，曾在 Kaggle、天池等国际算法大赛获得优异成绩：

- Kaggle Jane Street Market Prediction TOP1
- Kaggle Mechanisms of Action (MoA) Prediction 银牌
- Kaggle SIIIM-ISIC Melanoma Classification 银牌

本篇文章是明杰同学参加 Kaggle Jane Street Market Prediction 的竞赛总结，希望通过此次技术分享为各位同学提供些许算法思路，并在其他竞赛中取得更好成绩：)

#1

今天的贸易系统在很大程度上依赖于技术的运用。市场每天都在运作，为了在这种动态的市场中生存下去，我们需要使用所有有用的东西。机器学习模型就是一个十分出色的选择。因为它非常善于理解模式和预测，并且随着技术的发展，机器学习在市场价格预测中的应用越来越多。结合机器学习模型和人类知识可以做到十分出色的市场预测。

#2

在 Kaggle Jane Street Market Prediction 中，我们需要通过来自全球主要的证券市

在 Kaggle 的股票交易比赛中，我们从交易所的市场数据构建量化交易模型，并且在真实的未来数据上测试我们的模型性能。

该数据集包含一组匿名特征 `feature_{0...129}`，对于数据与标签我们都不知道它们代表了什么含义。我们只知道，它们代表真实的股票市场数据。

数据集中的每一行代表一个交易，我们将为其预测一个 `action` 值：1 表示进行交易，0 表示不进行交易。每笔交易都有一个关联的 `weight` 和 `resp`，它们组合在一起代表了交易的回报率。该 `date` 列是一个整数，代表交易日。

#3

比赛的 metrics 为 Utility Score：

$$p_i = \sum_j (weight_{ij} * resp_{ij} * action_{ij})$$

$$t = \frac{\sum p_i}{\sqrt{\sum p_i^2}} * \sqrt{\frac{250}{|i|}}$$

- `Pi` 代表了每一天的收入；
- `weight` 是购买 stock 的数量；
- `resp` 是未来交易时的价格浮动；
- `action` 则代表是否进行这次交易。

值得注意的是，比赛中存在很严重的过拟合公榜的现象，很多人发布了针对公榜调参的模型。最终在私榜上的抖动还是比较严重的，但是可以发现公榜和私榜存在线性相关。

简单数据格式如下：

date	weight	resp_1	resp_2	resp_3	resp_4	resp	feature_0	feature_1	feature_2	...
0	0	0.000000	0.009916	0.014079	0.008773	0.001390	0.008270	1	-1.872746	-2.191242
1	0	16.873515	-0.002828	-0.003226	-0.007319	-0.011114	-0.009792	-1	-1.349537	-1.704709
2	0	0.000000	0.025134	0.027607	0.033406	0.034380	0.023970	-1	0.812780	-0.256156
3	0	0.000000	-0.004730	-0.003273	-0.000461	-0.000476	-0.003200	-1	1.174378	0.344640
4	0	0.138531	0.001252	0.002165	-0.001215	-0.006219	-0.002604	1	-3.172026	-3.093182

我们的最终模型：XGB+MLP 的组合

● MLP 部分：

监督式自动编码器的方法最初是在这里

<https://www.kaggle.com/aimind/bottleneck-encoder-mlp-keras-tuner-8601c5> 提出的，其中一个监督式自动编码器在交叉验证 (CV) 拆分之前单独训练。但这种方法会引起一定程度的泄露，由于在最开始的时候编码器已经看到了全部的数据。所以需要在每个 CV 中单独训练一个自动编码器来减少泄漏。

● 交叉验证 (CV) 策略和特征工程：

5 折 purged group time-series 进行交叉验证

删除前 85 天进行培训（极度抖动且 CV 与 LB 都会下降）

前向填充缺失值（防止泄露）

将所有 `resp` 目标 (`resp`、`resp_1`、`resp_2`、`resp_3`、`resp_4`) 转移到用于多标签分类的 `action`

在推理过程中，我们使用了 `resp_3` 与平均值作为最终概率。详情见下节。

MLP 模型：

使用自动编码器创建新功能，将原始功能添加到 MLP

在每个 CV split 中一起训练自动编码器和 MLP 以减少泄漏



将 Target 添加到自动编码器（监督学习）以强制其生成更多相关特征。
在编码器之前添加高斯噪声层以进行数据增强并防止过度拟合
使用 Swish 激活函数代替 ReLU 来防止“死神经元”并平滑梯度
Batch Normalisation 和 Dropout 用于 MLP
用 3 个不同的随机种子训练模型并取平均值以减少预测方差
仅使用在最近两次 CV 拆分中训练的模型（具有不同种子），因为它们已经看到了更多数据
仅监控 MLP 的 BCE 损失，而不是提前停止的整体损失。
使用 Hyperopt 寻找最优超参数集

对于 XGB 模型，这里不再赘述。

之所以我们最终能取得一些非常高的分数，是因为我们针对于最终的 Target 发现了一些特殊的地方。简单来说：

官方在数据中提供了 5 组收益指标，分别是 resp_1, resp_2, resp_3, resp_4 和 resp。我们的最终收益是按照 resp 的值计算的。

根据 resp_1, resp_2 交易所取得的收益更加稳定，t 值高

根据 resp_3 交易所取得的收益有较高的 t 值，p 值也往往高

根据 resp, resp_4 交易可能取得很高的 p 值，但是 t 值往往很低

从以上结论我们发现了，利用 resp_3，我们可以得到既稳定，又有高收益的模型，也就是说如果我们在预测和训练的时候直接使用 resp_3，很可能得到比用 resp 预测更高的收益！

原因是因为比赛的指标 Utility Score：

与收入不同，utility score 还考察了模型收益的稳定性，即模型的最大回撤不能过大，不然的话就不是好模型，最终的 utility score 也会相对较低。可以看出：如果有某一天的收入为负的话，t 计算公式里的分母会变小，分子会变大，utility score 也会被惩罚得非常低。

另一个可以看出的要点是，在 t 值大于 6 的时候，总收入决定了 u 值的大小。

其他的一些实验结论

- 缺失值

如果训练时使用了多目标，取均值和众数取得的收益其实是差不多的，但是对于某些特殊情况，均值或者众数往往有奇效。鉴于这里的严重不确定性，我们没有在这里做过多的研究。

- 最佳的交易阈值

线下经过实验得到的结论是 0.51~0.52 之间的阈值可以带来更高的收益，最后的提交的模型中有一个是用 0.51 作为阈值。

- 最终模型

比赛的数据是匿名特征（130 列），特征工程不是完全无用，但是往往会吃力不讨好。为了保持解决方案的简洁性，最后提交的模型没有加入特征工程。

实际上有一些特征工程可以在线下取得分数的提升，但是加入线上模型会降低分数，我们这里希望只保留线上线下都有提升的强结论。

读到这里的读者可能会发现，我们并没有用什么特别复杂的方法。实际上我们的大部分的精力都花在了观察数据和模型的表现上，复杂的模型或者方法也尝试过，但是往往效果不好，线上线下不一致，或者并没有显著性。我们一致认为最终获胜的模型可能并不是那么复杂。

我们最终模型是 AE+MLP + XGBOOST(100 round)，每种模型各针对三个种子进行了训练。截至 2021-07-06。我们目前的成绩是 TOP1。



█ | < < 1/19 > >|