基于 DNN、GBT、RF 构建的标普 500 统计套利策略

赵浩翰
2021 级商学院金融学

日期：2024 年 8 月 20 日

摘　　要

本文是对 Krauss, Xuan and Nicolas (2017)[1] 一文的复现，我们实现并分析了 DNN、GBT、RF 等机器学习模型及其相关集成模型在统计套利背景下的有效性。各个模型均在标普 500 成分股上构建和测试。除原文内容外，我们将数据拓展至 2024 年，且实现了 XGBOOST、LightGBM 模型作为补充，并使用网格搜索对各个模型的参数进行了调优。

本文的复现由我和赵润平共同完成，我主要负责数据获取、数据处理、特征生成、模型构建和训练部分，因此我的报告主要介绍以上内容，策略回测和表现分析请查看另一份报告。

关键词： 机器学习，统计套利

1　背景介绍

一般而言，统计套利是指预测股票胜过整体市场的概率，由此产生交易信号，买入被低估的股票而卖出被低估的股票。本文实现并分析了诸多先进的机器学习模型在统计套利中的有效性，包括 RF、GBDT、XGBOOST、LightGBM 等树模型以及 DNN 深度神经网络模型，各个模型都基于最新的 S&P 500 日频数据进行训练，并经过网格搜索进行参数调优。

我们详细完成了数据收集、数据清洗和整理、模型训练、结果测试和分析等内容，其中，我主要负责了数据获取、数据处理、特征生成、模型构建和训练部分，因此我的报告主要介绍以上内容。本文的第二部分介绍了数据来源和使用的软件，第三部分介绍了数据处理和模型训练的方法论，第四部分分析了模型训练的结果，第五部分着重展示了模型改进，第六部分对我的成果进行了总结。

2　数据与软件

综合考虑数据的可获得性、计算的可行性、市场的有效性、流动性等因素，我们选择了 S&P 500 作为本文的股票域，数据的获取及初步处理细节如下：

1. 获取 S&P 500 自 1996 年 12 月至 2024 年 3 月 31 日的月频成分股、各股票对应的行业以及历史变动数据，取自 GitHub repository。
2. 在每个月末获取指数成分股列表，转换为一个二项矩阵，表示各股票下个月是否是指数的成分股。
3. 依据 Krauss and Stübinger (2015) 消除幸存者偏差。
4. 对于所有曾是指数成分股的股票，获取其 1996 年 1 月至 2024 年 4 月的日频股票数据，数据源为 Yahoo Finance。在 Python 中对应的第三方库名为 yfinance。

我们一共得到了 1163 只股票，去除了已经退市或无法获取数据，以及有效交易日小于 1000 天的股票后，最后剩余 791 只。

由于没有公开且免费的官方成分股历史数据源，我们使用了许多不同的数据库，但由于各个数据库的维护不一，对退市、更名等没有有效处理，我们已尽可能保证原始数据的质量，并尽力清洗数据以保持其合理性，但仍难以避免与原论文中的差异。

考虑到各种机器学习方法的实现和数据处理的便捷性，我们选择了 Python 作为实现语言。数据处理主要基于 Pandas 和 Numpy，DNN 神经网络基于 Pytorch，RF 和 GBDT 主要基于 sklearn，XGBoost 和 LightGBM 均基于其同名 Python 第三方库 xgboost 和 lightgbm。

3 方法论

3.1 划分训练集与数据集

本文的模型训练基于“学习期”，我们将一个“学习期”定义为一个训练-测试集 (也称为交易期)，包括 750 天的训练期 (约 3 年) 和其后 250 天的测试期 (约 1 年)，共 1000 天。我们使用滚动窗口，以 250 天为步长来划分学习期，由此得到连续但不重叠的交易期。

我们收集到的数据共有 7130 个交易日，考虑到步长为 250 天，所以我们先去掉前 130 天，因此，实际上我们的训练数据从 1996 年 7 月 5 日开始。之后以剩余的 7000 天进行学习期的划分，共得到 25 个学习期。

在每个学习期内，我们使用最初的 240 天进行特征的计算，最后的 250 天作为交易期，所以我们去除每个学习期内有效交易天数小于 600 天的股票，最后，每个学习期内股票的数量如图 1：

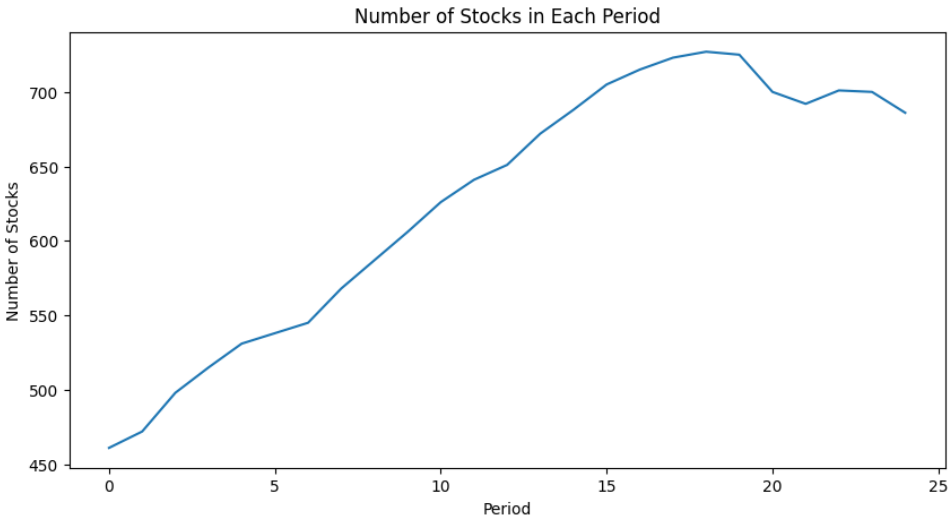

图 1: 各学习期股票数量

3.2 特征计算

我们使用的特征与原文一致，在每个学习期内分别生成特征空间 (输入) 和响应变量 (目标)，具体生成方式如下。

对于输入的特征空间，记 $P^s = (P^s_t)_{t\in T}, s \in \{1,\cdots,n\}$ 为股票 s 的价格序列，$R^s_{t,m}$ 为股票 s 过去 m 天的对数收益率，由此得到 31 个特征，此时会失去学习期最初的 240 个交易日。$R^s_{t,m}$ 计算方式如式 (1)：

$$R^s_{t,m} = \log\left(\frac{P^s_t}{P^s_{t-m}}\right), \quad m \in \{\{1,\cdots,20\} \cap \{40,60,\cdots,240\}\} \tag{1}$$

对于输出的标签，任意股票 s，我们定义一个二项变量 $Y^s_{t+1} \in \{0,1\}$，当 $R^s_{t+1,1}$ 大于市场收益率中位数时，$Y^s_{t+1} = 1$，反之则为 0。

如上所述，我们的目的就是建立模型，对于任意股票 s，依据 t 日的特征，预测一个概率 $\mathcal{P}^s_{t+1|t}$，表示其 $t+1$ 日的表现胜过市场横截面中位数的概率。

综上，在任意一个学习期内，我们有 510 天训练集，250 天交易集，训练集的大小为：$510 \times n_i \times 32$，交易集的大小为：$250 \times n_i \times 31$，其中，n_i 是第 i ($i \in \{1, 2, \cdots, 25\}$) 个学习期的股票数量。

3.3 模型构建

3.3.1 深度神经网络

在计算机性能日渐强大的信息时代，神经网络，这一拥有大量参数、可拟合复杂非线性关系的强劲模型受到了广泛关注。典型的深度神经网络的拓扑结构一般为：一个输入层、一个或多个隐层、以及一个输出层。输入层与特征空间相匹配，其神经元个数与特征数相同，可以将原始特征映射到更高维的空间。隐层之间彼此相连，以激活函数作为中介，实现复杂的特征变换。输出层则与输出空间相匹配，根据问题的类型，如回归或分类，将隐层的输出变换为我们需要的输出。所有的神经层都由神经元构成，在神经元内部进行线性加权组合变换，如对于 l 层的 n_l 个输出，有：

$$\alpha = \sum_{i=1}^{n_l} w_i x_i + b \tag{2}$$

下面详述我们构建的深度神经网络模型。首先，网络的拓扑结构为：I–H1–H2–H3–O，各有 31-31-10-5-2 个神经元，故共有 2746 个参数，按训练集 $510 \times 500 \times 32$ 来估计，每个参数可以对应 93 个训练样本，因此各个参数可以得到充分的估计。

对于隐藏层的激活函数，我们选择了 MAXOUT 激活函数，其计算方式如式 (3) 所示：

$$\mathrm{f}(\alpha_1, \alpha_2) = \max(\alpha_1, \alpha_2), \quad \mathrm{f} : \mathbb{R}^2 \to \mathbb{R} \tag{3}$$

在输出层，由于我们的问题属于二分类，所以我们选择了 Sigmoid 激活函数，其计算方式如式 (4) 所示：

$$sigmod(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

同样，由于我们的问题属于二分类问题，我们选择了 Cross-Entropy 损失函数，记 W, B 为所有的权重矩阵和偏置，则 Cross-Entropy 可以如式 (5) 计算：

$$\mathcal{L}(W, B|j) = -\sum_{y \in O} \left(ln(o_y^{(j)}) t_y^{(j)} + ln(1 - o_y^{(j)})(1 - t_y^{(j)}) \right) \tag{5}$$

关于优化器，我们选择了 Adam，它是 Pytorch 中常用的随机梯度下降优化器，在处理非稳定目标时表现较好，从而得到了广泛使用。我们设置学习率为 0.001。

考虑到过拟合风险，我们为输入层和隐层添加了概率为 0.1 和 0.5 的 Dropout 层，并设置了 L_1 正则化，惩罚系数 $\lambda_{DNN} = 10^{-5}$。

最后，我们将训练轮次设置为 400，且采用早停逻辑：当训练损失超过 20 个轮次没有下降时，则提前停止训练。

3.3.2 随机森林

树模型是机器学习中的一大类模型，最基础的决策树模型可以模拟人的判断，生成一个树状的决策路径，通过不断学习使这个决策过程更加准确。从基础的决策树模型出发，发展出了许多更加复杂、先进的模型，接下来描述的随机森林、梯度提升树、XGBoost 和 LightGBM 均是树模型家族的成员。

机器学习中，“集成”是一个非常长重要的思想，它可以将一些弱学习器组合到一起，形成更强大的模型。而集成又主要包括两种方式：boosting 和 bagging。随机森林便是 bagging 集成学习的典型代表。简言之，随机森林将许多弱学习器，即决策树，“并列”地组合到一起，由众多基学习器独立判断和决策、并最终“投票”得到预测结果，由此提升模型的预测能力。

随机森林的各个基学习器可以随机抽样、随机选择特征等，进行独立的演化，生成许多棵独立的深度决策树，通过大量树的叠加来避免过拟合和增强泛化能力。一个简单的示例如图 2 所示：

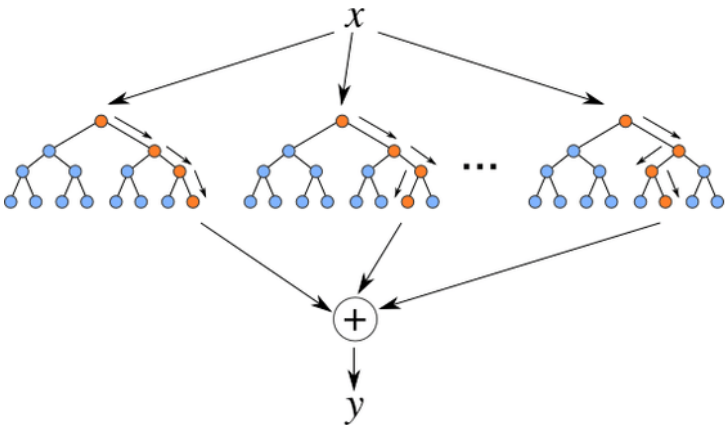

图 2: 随机森林

我们的随机森林模型使用 Python 第三方库 `sklearn.ensemble.RandomForestClassifier` 实现，其主要参数经过 `sklearn.model_selection.GridSearchCV` 进行选择，参数如下：

- 基学习器数量 (树的数量)：$B_{RAF} = 1000$。
- 单棵树的深度：$J_{RAF} = 20$。
- 特征子集的个数：$m_{RAF} = \lfloor \sqrt{p} \rfloor$。

3.3.3 梯度提升树

与随机森林相同，梯度提升树也是一个典型的集成树模型，但它采用的集成学习方法为 boosting，即将一系列弱学习器组成序列，每个学习器都聚焦于前辈们在训练样例上的偏差，并致力于减小训练集上的损失，从而生成一串不断增强的模型，实现最终预测效果的增强。简言之，bagging 的思想更像电路的“并联”，各个基学习器独立地发生作用，而 boosting 则更像“串联”，基学习器前后联系，基于前序学习器的结果发挥作用。

梯度提升树的算法可以描述如下：

1. 初始化 $f_0(x) = 0$，设损失函数为 $L(y, f(x))$。
2. 对于 $m = 1, 2, \cdots, M$：
   (a) 计算损失函数的负梯度：$-g_m(x_i) = -\left[\frac{\partial L(y, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$。
   (b) 以负梯度 $-g_m(x)$ 为目标训练一棵回归树 $T_m(x, \Theta)$。
   (c) 得到第 m 轮的模型 $f_m(x) = f_{m-1}(x) + \rho T_m(x, \Theta)$。
3. 在 M 轮后得到最终的模型：M iterations: $\mathrm{f}_M(x) = \sum_{m=1}^{M} \rho T_m(x, \Theta)$。

我们的梯度提升树使用了 Python 第三方库 `sklearn.ensemble.GradientBoostingClassifier` 实现，其主要参数经过 `sklearn.model_selection.GridSearchCV` 进行选择，参数如下：

- 基学习器数量：$M_{GBT} = 100$。
- 树的深度：$J_{GBT} = 3$。
- 学习率：$\lambda_{GBT} = 0.1$。
- 特征子集个数：$m_{GBT} = 15$。

4 结果与分析

我主要负责了数据处理、模型构建和模型训练部分，数据处理和模型构建部分的结果在上文已有阐述，下面仅阐述模型训练的结果部分。

首先，简单介绍各个模型的训练流程，按照之前所述，我们将所有的数据划分为 25 个长为 1000 天的学习期，并使用每个学习期内的前 240 天计算特征值，之后使用 510 天作为训练数据，并使用最后的 250 天作为交易集。

在每个学习期内，我们定义需要训练的模型为 M，其参数集为 Θ，则单个学习期内的训练和交易过程可以描述如下：

1. 准备数据集和特征，划分为训练集和交易集。
2. 初始化模型 M，使用 `Python` 的第三方库进行。
3. 初始化需要搜索的参数列表。
4. 使用 `sklearn.model_selection.GridSearchCV` 搜索参数。
5. 使用训练集的最优参数拟合模型，进行交易集上的测试。
6. 计算准确度 Accuracy、精确度 Precision、召回率 Recall、F1、AUC 以及基尼系数，以评估实际交易的表现。

最终，将各个模型在每个学习期的表现汇总，得到的结果描述如下，图 3、4、5 分别展示了各个模型在所有学习期内各指标的平均值、最大值和最小值。

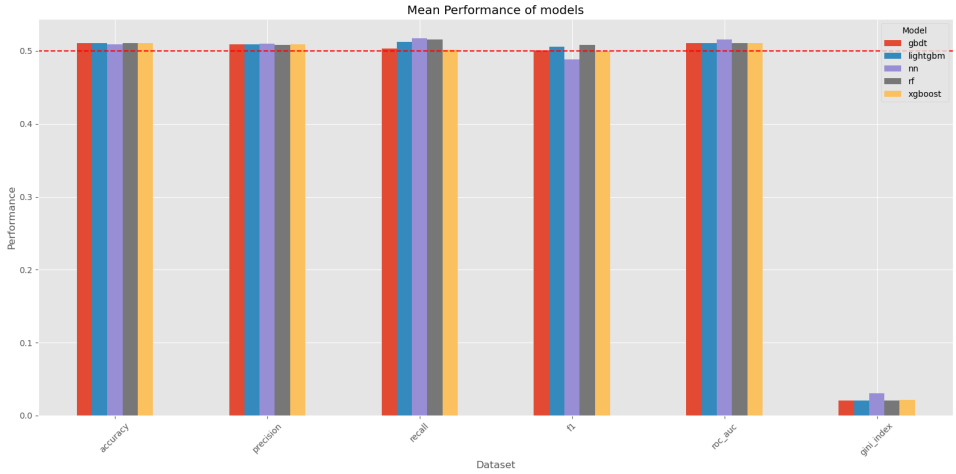

图 3: 交易集指标平均值

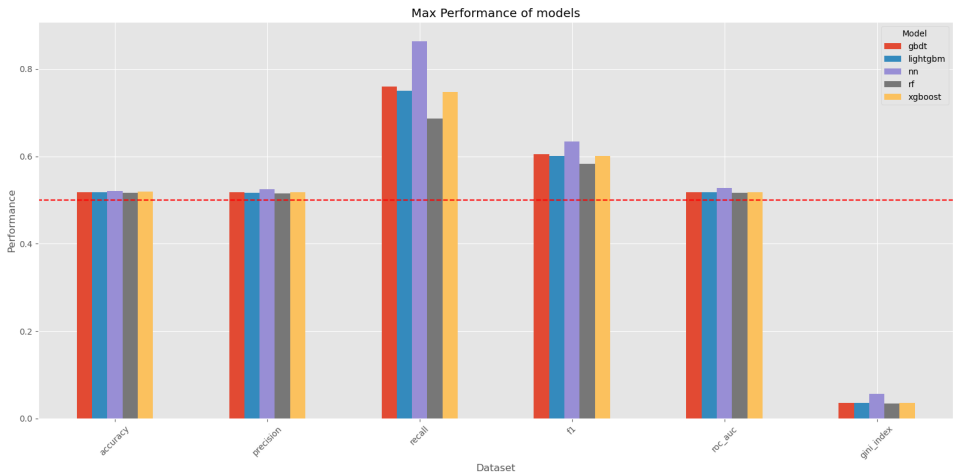

图 4: 交易集指标最大值

从图中可以看到，各个模型的的综合表现较好，且各模型之间的表现差别不大，DNN 在部分指标中的表现方差较大，其他模型的表现则相对稳定。训练效果整体较好。

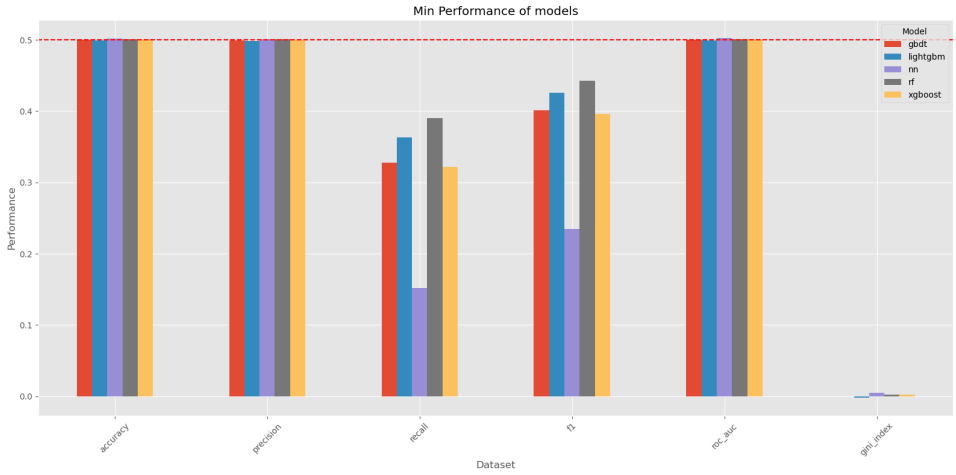

图 5: 交易集指标最小值

除此之外，我对各个模型进行了特征重要性分析，其中，树模型可以依据分叉过程自然地得到特征重要性，而 DNN 则以第一层神经网络的权重值评估特征的重要性。最后，将特征重要性进行 MinMax 标准化，取排名最高的前 30 个特征展示如下：

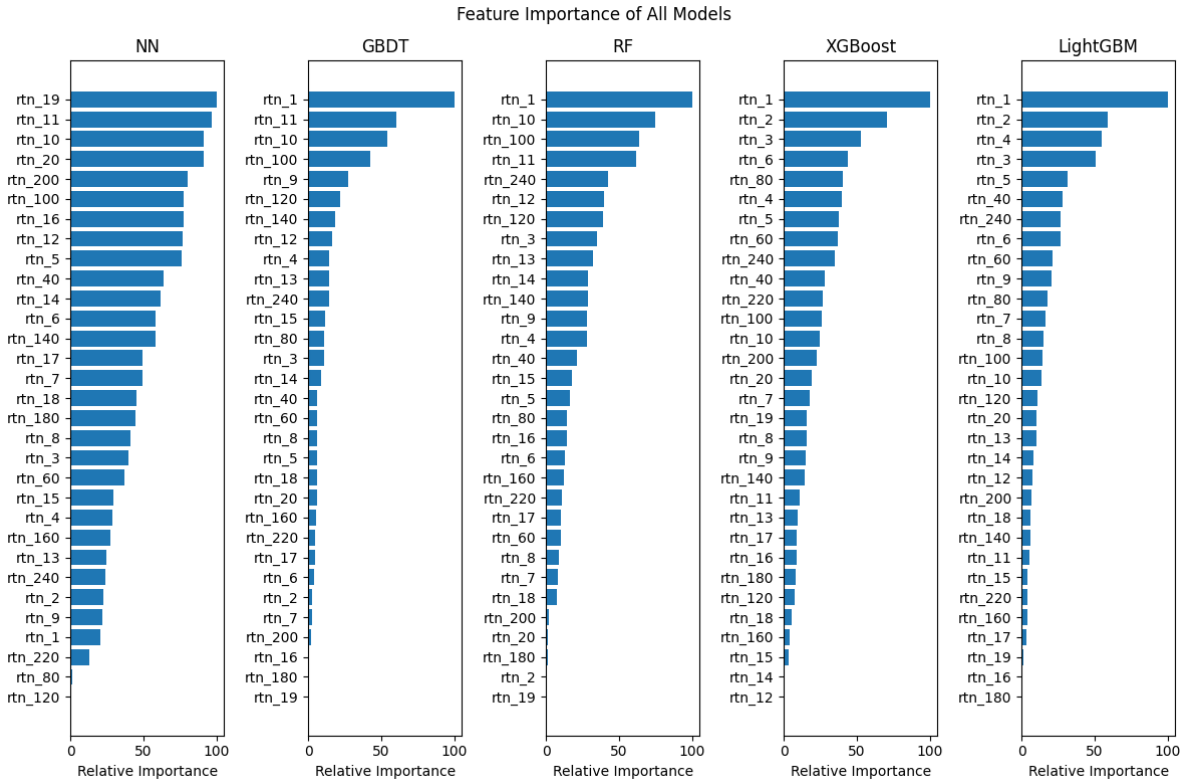

图 6: 各模型特征重要性

如图 6 所示，总体而言，在各个数模型中，过去较短时间内的对数收益率特征相对重要，即 $rtn_1 - rtn_{20}$ 的特征出现频率较高，除此之外，不分具有特殊含义的区间，如一个月 rtn_{20}、一年 rtn_{240} 等特征也相对重要。

5 改进

我们做的改进包括两部分：新增模型和优化模型训练过程。前者包括 XGBoost 和 LightGBM，后者主要是使用了交叉验证网格搜索来确定模型参数。

5.1 新增模型

考虑到近年树模型的更新发展，在 Boosting 树模型中，产生了两个较有代表性的模型 XGBoost、LightGBM，我们便考虑将 GBDT 替换为这两个更先进的模型，以检验套利策略的效果是否可以得到提升。由于模型的训练过程与之前的模型类似，下面仅简述两个模型的构建细节以及使用的工具。

5.1.1 XGBoost

在机器学习中，偏差和方差是两个常见的问题，分别度量模型的准确性及稳定性。XGBoost 是 GBDT 的改进版本，通过**添加损失函数的二阶导数**来提高模型的准确性，并通过**对模型参数施加惩罚**来减少模型的方差。具体可以以其目标函数来进行解释，XGBoost 的目标函数可以表示如下：

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \\
where \quad & l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) = l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)
\end{aligned}
\tag{6}
$$

目标函数是 XGBoost 和 GBDT 之间最重要的区别，也是使其在偏差和方差上表现更好的原因。一方面如式 (6) 所示，XGBoost 在目标函数的 Loss 项中加入了 $\frac{1}{2} h_i f_t^2(x_i)$，即 Loss 的二阶泰勒展开项，从而使其对原函数的近似更加精确，减小偏差。另一方面，XGBoost 在目标函数中加入了 $\sum_{i=1}^{t} \Omega(f_i)$ 项，从而为模型参数施加惩罚，以防止过拟合，减小方差。

XGBoost 还使用许多其他方法，如**列块并行学习**、特殊的**最优分割算法**等来提高训练效率。

模型的训练使用 Python 的 xgboost 包，仍使用 `sklearn.model_selection.GridSearchCV` 搜索参数。模型主要的训练代码如下所示：

```
xgb_params_grid = {
    "n_estimators": [100, 200, 300],
    "max_depth": [3, 5, 7],
    "learning_rate": [0.01, 0.1, 0.3],
    "subsample": [0.5, 0.8, 1],
    "colsample_bytree": [0.5, 0.8, 1],
}

xgb_params = {
    'booster': 'gbtree',
    'objective': "reg:squarederror",
    'lambda': 0.1, # L2 regularization term on weights
    'min_child_weight': 2,
    'nthread': 8, # Number of threads
}

xgb_model = xgb.XGBRegressor(**xgb_params)
with timer("Grid Search XGBoost"):
    xgb_grid_search = GridSearchCV(
        xgb_model, param_grid=xgb_params_grid, cv=5, n_jobs=-1, scoring=rmspe_scorer
```

```
    )
    xgb_grid_search.fit(X_train_scaled, y_train)

print("Best parameters found: ", xgb_grid_search.best_params_)
print("Lowest RMSPE found: ", xgb_grid_search.best_score_)
```

5.1.2 LightGBM

LightGBM 是 XGBoost 的改进版本，它在处理大规模数据集时因其高效的训练效率和仍然保持的高准确性而表现更好。

XGBoost 的整体训练复杂性可以大致估计为：

$$\text{树的数量} \times \text{每棵树的叶子数} \times \text{生成每片叶子的复杂度}$$

由于 XGBoost 采用的基模型是二叉树，每个叶子需要分裂一次来生成。对于每次分裂，需要遍历所有特征上的所有候选分割点来计算最大改进。因此，生成一个叶节点的复杂性可以估计为：

$$\text{特征数量} \times \text{候选分割点数量} \times \text{样本数量}$$

LightGBM 主要使用三种方法来减少生成叶节点的复杂性：**直方图算法 (Histogram)** 减少候选分割点数量，**基于梯度的单边采样 (GOSS)** 减少样本数量，以及**互斥特征捆绑 (EFB)** 减少特征数量。

- **直方图算法：**通过将连续特征离散化为固定数量的箱，如 255，来减少候选分割点数量至 $N_{\text{bin}} - 1$。
- **基于梯度的单边采样：**保留绝对梯度值较大的样本，同时按一定比例采样梯度值较小的样本。
- **互斥特征捆绑：**将大部分值为 0 的稀疏特征 (被认为彼此是互斥的) 捆绑在一起。

模型的训练使用 Python 的 lightgbm 包，仍使用 `sklearn.model_selection.GridSearchCV` 搜索参数。模型主要的训练代码如下：

```
lgb_params_grid = {
    "n_estimators": [100, 200, 300],
    "max_depth": [3, 5, 6, 7],
    "learning_rate": [0.01, 0.025, 0.05, 0.1],
    'subsample': [0.5, 0.7, 1],
    'colsample_bytree': [0.5, 0.7, 1],
}
lgb_params = {
    'objective': 'regression',
    'boosting_type': 'gbdt',
    'lambda_l2': 0.1,
    'min_child_weight': 2,
    'nthread': 8,
    'verbose': -1,
}

lgb_model = lgb.LGBMRegressor(**lgb_params)
with timer("Grid Search LightGBM"):
    lgb_grid_search = GridSearchCV(
        lgb_model, param_grid=lgb_params_grid, cv=5, n_jobs=-1, scoring=rmspe_scorer, verbose
            =1
    )
    lgb_grid_search.fit(X_train, y_train)

print("Best parameters found: ", lgb_grid_search.best_params_)
print("Lowest RMSE found: ", np.sqrt(np.abs(lgb_grid_search.best_score_)))
```

5.2 交叉验证网格搜索

原文中，各个模型的参数都是直接设定的，并未经过参数调优，我们对这一环节进行了改进，每个学习期内，在训练集上训练时，使用 `sklearn.model_selection.GridSearchCV` 通过 5 折交叉验证搜索最佳参数，使用最佳参数拟合新的模型后，再运用至交易集进行交易。具体使用方式可以参考如上 lightgbm 的训练，即：

```
with timer("Grid Search LightGBM"):
    lgb_grid_search = GridSearchCV(
        lgb_model, param_grid=lgb_params_grid, cv=5, n_jobs=-1, scoring=rmspe_scorer, verbose
            =1
    )
    lgb_grid_search.fit(X_train, y_train)

print("Best parameters found: ", lgb_grid_search.best_params_)
print("Lowest RMSE found: ", np.sqrt(np.abs(lgb_grid_search.best_score_)))
```

6 总结

本篇报告关注于论文复现中数据获取、数据处理、特征生成、模型构建和训练等环节，在完成原文工作的基础上，进行了进一步的探索和改进。

数据方面，我们仍使用原文的 S&P 500 股票域，采取相同的逻辑整理和处理数据，并生成特征空间，尽我们所能保证了数据质量，并将数据更新至 2024 年，以探索策略近年来的表现。

模型构建方面，我们使用 Python 及其第三方库完成了模型的构建和训练，实现了各个学习期内的训练、预测和交易，并完成模型表现评估和特征重要性分析，且加入交叉验证网格搜索环节以寻找各个模型的最优参数。同时，考虑到树模型近年来的新发展，我们将 XGBoost 和 LightGBM 加入套利策略之中，以求改进策略表现。

本文详细描述了我负责的各个环节的实现细节和最终结果，关于套利策略最终的回测结果和表现分析，请参考另一篇报告。

参考文献

[1] Christopher Krauss, Xuan Anh Do, and Nicolas Huck. "Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500". In: *Eur. J. Oper. Res.* 259 (2017), pp. 689–702. URL: https://api.semanticscholar.org/CorpusID:207747899.