Python 量化交易：用递归量化分析（RQA）预测市场走势

原创 数据科学实战 数据科学实战 2025年09月08日 00:00

引言

在量化金融领域，预测市场方向一直是核心挑战。传统的技术指标（如移动平均线、RSI、布林带）虽然提供了有用的信号，但往往受限于其线性特性。而金融市场本质上是非线性且复杂的系统。

那么问题来了：我们能否借助动力系统理论中的非线性方法来提取新的预测信息？

本文将探索基于递归量化分析（Recurrence Quantification Analysis，RQA）的市场预测方法。我们将通过 Python 实现一个完整的分析流程，包括：

- 获取加密货币价格数据
- 从滚动时间窗口的收益率计算 RQA 特征
- 使用逻辑回归预测次日市场方向
- 通过前向步进测试和噪声扰动评估模型稳健性
- 将结果与买入持有策略进行比较

让我们一起深入了解 RQA 指标背后的理论以及在交易研究中应用它们的实用工作流程。

RQA 的基本概念

递归量化分析（RQA）是非线性动力学中的一种技术，用于分析相空间中系统的递归结构。通过将时间序列嵌入到更高维度，我们可以研究重复模式、持久性和可预测性。

RQA 的关键指标包括：

- **递归率（RR）**：重复状态的频率
- **确定性（DET）**：可预测对角线结构的比率
- **层状性（LAM）**：停滞状态的普遍性
- **最大线长（Lmax）**：最长对角线，与可预测性范围相关
- **熵（ENTR）**：递归结构的复杂性

这些指标将非线性结构转化为适合机器学习的量化特征。

实现步骤

1. 导入必要的库

首先，我们需要导入各种用于数据处理、机器学习和可视化的库：

```
import numpy as np
import pandas as pd
import yfinance as yf
from scipy.spatial.distance import pdist, squareform
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score
from dataclasses import dataclass
import ccxt
import matplotlib.pyplot as plt
```

2. 数据获取

我们定义一个辅助函数从 Binance 获取 OHLCV 蜡烛图数据：

```
def fetch_binance(symbol="BTC/USDT", timeframe="1d", since="2018-01-01T00:00:00Z", limit=100
    exchange = ccxt.binance()
    since_ms = exchange.parse8601(since)
    all_candles = []

    while True:
        candles = exchange.fetch_ohlcv(symbol, timeframe=timeframe, since=since_ms, limit=li
        if not candles:
            break
        all_candles.extend(candles)
        since_ms = candles[-1][0] + 1
        if len(candles) < limit:
            break

    # 创建包含日常 OHLCV 数据的 DataFrame
    df = pd.DataFrame(all_candles, columns=["timestamp", "open", "high", "low", "close", "vo
    df["timestamp"] = pd.to_datetime(df["timestamp"], unit="ms")
    df.set_index("timestamp", inplace=True)
    return df
```

3. RQA 指标计算

以下是计算 RQA 指标的核心函数：

```
def embed(series, m, tau):
    """
    将时间序列嵌入到高维空间

    参数:
    series: 原始时间序列
    m: 嵌入维度
    tau: 时间延迟
    """
    x = np.asarray(series, float)
    n = x.size
    L = n - (m - 1) * tau
    if L <= 2 or m < 2 or tau < 1:
        return None
    idx = np.arange(L)[:, None] + np.arange(m)[None, :] * tau
    return x[idx]

def recmat(emb, eps_q=0.1):
    """
    计算递归矩阵

    参数:
    emb: 嵌入的时间序列
    eps_q: 距离阈值（分位数）
    """
    d = pdist(emb, metric='euclidean')
    eps = np.quantile(d, eps_q)
    R = squareform(d) <= eps
    np.fill_diagonal(R, False)
    return R

def rqa_metrics(series, m=5, tau=1, eps_q=0.1):
    """
    计算 RQA 指标

    参数:
    series: 原始时间序列
    m: 嵌入维度
    tau: 时间延迟
    eps_q: 距离阈值（分位数）
    """
    emb = embed(series, m, tau)
    R = recmat(emb, eps_q)
    N = R.size

    # 计算各种 RQA 指标
    rr = R.sum() / N  # 递归率

    # 这里假设 _rle_lengths 和 _vert_lengths 函数已定义
    dls = [len(run) for k in range(-(R.shape[0]-1), R.shape[0])
           for run in _rle_lengths(np.diagonal(R, offset=k)) if len(run)>1]
    vls = _vert_lengths(R)

    det = (np.sum(dls) / R.sum()) if dls else 0  # 确定性
    lam = (np.sum(vls) / R.sum()) if vls else 0  # 层状性
    lmax = max(dls) if dls else 0  # 最大线长
    entr = _entropy(dls)  # 熵

    return dict(RR=rr, DET=det, LAM=lam, Lmax=lmax, ENTR=entr)
```

4. 滚动特征计算

我们在收益率上计算滚动 RQA 特征：

```
def rolling_rqa(returns, lookback=200, m=5, tau=1, eps_q=0.1):
    """
    计算滚动 RQA 特征

    参数:
    returns: 收益率序列
    lookback: 回溯窗口大小
    m: 嵌入维度
    tau: 时间延迟
    eps_q: 距离阈值（分位数）
    """
    feats = np.full((len(returns), 5), np.nan)
    for t in range(lookback, len(returns)):
        window = returns[t - lookback:t]
        mtr = rqa_metrics(window, m, tau, eps_q)
        feats[t] = [mtr['RR'], mtr['DET'], mtr['LAM'], mtr['Lmax'], mtr['ENTR']]
    return pd.DataFrame(feats, columns=['RR', 'DET', 'LAM', 'Lmax', 'ENTR'])
```

5. 前向步进评估

我们实现一个滚动的训练/测试循环：

```
@dataclass
class WFConfig:
    """前向步进配置类"""
    train: int = 365  # 训练窗口天数
    test: int = 90    # 测试窗口天数
    step: int = 30    # 步进天数

def walk_forward(X, y, cfg=WFConfig()):
    """
    执行前向步进测试

    参数:
    X: 特征矩阵
    y: 目标变量（明日方向）
    cfg: 配置参数
    """
    accs, aucs = [], []
    n = len(y)
    start = cfg.train

    for t0 in range(start, n - cfg.test, cfg.step):
        # 定义训练和测试区间
        tr, te = slice(t0 - cfg.train, t0), slice(t0, t0 + cfg.test)

        # 标准化特征
        sc = StandardScaler().fit(X[tr])
        Xtr, Xte = sc.transform(X[tr]), sc.transform(X[te])

        # 训练逻辑回归模型
        clf = LogisticRegression(max_iter=200).fit(Xtr, y[tr])

        # 预测并评估
        prob = clf.predict_proba(Xte)[:, 1]
        pred = (prob >= 0.5).astype(int)
        accs.append(accuracy_score(y[te], pred))
        aucs.append(roc_auc_score(y[te], prob))

    return dict(acc=np.mean(accs), auc=np.mean(aucs))
```

6. 实验执行

我们对嵌入维度（m）和延迟（τ）进行网格搜索，添加噪声稳健性检查，并总结结果：

```
if __name__ == '__main__':
    # 运行实验
    df, best = run_experiment(
        ticker='ETH/USDC',
        start='2023-01-01',
        m_list=(3, 7, 10),
        tau_list=(1, 3, 5),
        lookback=90,
        eps_q=0.1,
        noise_levels=(0.0, 0.25, 0.5, 1.0),
        seeds=(0, 1, 2, 3, 4)
    )

    # 输出报告和可视化
    console_report(df)
    plot_auc_heatmaps(df)
    plot_noise_curves(df)
    plot_stability(df)
```

7. 策略回测与可视化

我们将预测转换为多头/空仓或多头/空头策略，并与买入持有策略进行比较：

```
# 生成权益曲线
eq_lf = make_equity_curves(px, ret, pred_df, threshold=0.5, mode='long_flat')
eq_ls = make_equity_curves(px, ret, pred_df, threshold=0.5, mode='long_short')

# 绘制策略对比图
plt.figure(figsize=(12, 6))
plt.plot(eq_lf['eq_strat'], label="策略（多头/空仓）")
plt.plot(eq_lf['eq_bh'], label="买入持有")
plt.legend()
plt.title("RQA 策略与买入持有比较")
plt.show()
```

实验结果

在 ETH/USDC（2023-2024）上的样本输出：

- 最佳配置（m=10，τ=1）的 AUC 约为 0.53
- 策略相比买入持有改善了风险调整后的表现
- 多头/空头扩展进一步提高了收益，但增加了波动性

```
[[最稳定配置]] m=10, τ=1, noise=0.0
多头/空仓统计: {'年化收益': '0.392', '年化波动率': '0.352', '夏普比率': '1.114'}
买入持有统计: {'年化收益': '0.284', '年化波动率': '0.525', '夏普比率': '0.541'}
多头/空头统计: {'年化收益': '0.516', '年化波动率': '0.524', '夏普比率': '0.985'}
```

这些结果表明 RQA 特征具有适度的预测能力，并且与被动持有相比，能够显著降低风险。

应用与扩展

本方法可以进一步扩展和应用：

1. **资产范围**：应用于股票、外汇或商品
2. **风险管理**：添加止损、仓位调整和资本分配
3. **建模方法**：用随机森林、梯度提升或神经网络替代逻辑回归
4. **特征工程**：将 RQA 与波动率、动量或订单簿特征结合
5. **超参数优化**：使用贝叶斯搜索而非网格搜索来优化(m, τ, eps_q)
6. **投资组合构建**：跨多资产测试以获取多样化收益

总结

本研究展示了递归量化分析（RQA）如何集成到前向步进交易框架中。关键要点：

1. RQA 提供了能够捕捉价格序列中隐藏动态的非线性结构特征
2. 基于 RQA 特征的逻辑回归显示出优于随机猜测的预测优势
3. 前向步进测试突显了模型的稳健性并降低了过拟合风险
4. 策略权益曲线表明相较于买入持有风险调整后的改进

虽然 RQA 不是万能的解决方案，但它为量化工具包添加了有价值的非线性动态信息。通过进一步改进，它可以在对冲基金环境中补充现有的 alpha 信号。

好文推荐

1. 用 Python 打造股票预测系统：Transformer 模型教程（一）
2. 用 Python 打造股票预测系统：Transformer 模型教程（二）
3. 用 Python 打造股票预测系统：Transformer 模型教程（三）
4. 用 Python 打造股票预测系统：Transformer 模型教程（完结）
5. 揭秘隐马尔可夫模型：因子投资的制胜武器
6. YOLO 也能预测股市涨跌？计算机视觉在股票市场预测中的应用
7. 金融 AI 助手：FinGPT 让你轻松掌握市场分析
8. 量化交易秘籍：为什么专业交易员都在用对数收益率？
9. Python 量化投资利器：Ridge、Lasso 和 Elastic Net 回归详解
10. 掌握金融波动率模型：完整 Python 实现指南

量化投研 · 目录
上一篇
PCA 动量策略：如何通过主成分个数（K）优化交易信号
下一篇
双动量交易策略实战：从股票ETF到比特币的全资产应用