高斯过程回归(Gaussian Processes for Regression)预测股票

——随机过程期末作业报告

学号：2153592

姓名：胡逸凡

1.项目描述与现实背景

1.1.项目描述

股票市场是一个充满风险和不确定性的金融市场，对于投资者和交易者来说，准确预测股票价格的走势至关重要。然而，股票价格受到众多因素的影响，包括经济指标、公司财务状况、市场情绪等，这使得股票价格的预测变得非常具有挑战性。

在这个背景下，高斯过程回归是一种广泛应用的预测方法。它是一种非参数的贝叶斯回归方法，通过基于先验假设的高斯过程建模，可以对股票价格的未来走势进行预测。相比于传统的回归方法，高斯过程回归具有更强的灵活性和鲁棒性，并且能够提供对预测结果的不确定性估计。

本项目旨在利用高斯过程回归模型来预测股票价格的变化趋势，为投资者提供决策支持和交易策略。项目的目标是建立一个可靠的预测模型，能够在给定股票历史价格和相关因素的情况下，对未来的股票价格进行预测，并提供相应的置信区间。

我作为大数据专业并辅修金融学的学生对此项目也十分感兴趣。

1.2.项目价值与意义

通过高斯过程回归预测股票价格，可以帮助投资者更好地理解市场趋势，降低投资风险，提高投资收益。该项目的成功将有助于投资者制定更准确的投资策略，优化资产配置，并在股票市场中获取长期竞争优势。此外，高斯过程回归方法还可应用于其他金融市场和领域，如期货市场、外汇市场等，具有广泛的应用前景和研究意义。通过深入研究和应用高斯过程回归，可以为金融领域的预测与决策提供新的思路和方法。

1.3.GPR的介绍

与其他回归方法相比，GPR拥有以下**优点**：

预测的不确定性：高斯过程回归的一个重要特点是能提供预测结果的不确定性度量。在许多实际应用中，除了得到预测值之外，我们还需要知道预测的不确定性。例如，如果我们对气候模型进行预测，我们不仅需要知道预测的平均温度，而且还需要知道这个预测的不确定性范围。

非参数：高斯过程是一种非参数的方法，这意味着它可以适应数据的复杂性，而不是假定数据遵循特定的分布或函数形式。这使得高斯过程回归在处理复杂数据时非常灵活。

核函数的灵活性：通过使用不同的核函数，高斯过程回归可以捕获数据中的各种各样的依赖关系和模式。这使得高斯过程回归能够适应各种不同类型的数据和问题。

贝叶斯方法：高斯过程是一种贝叶斯方法，这意味着它可以结合先验知识和观测数据进行预测。这对于处理有限的数据或是具有强烈先验信念的问题非常有用。

然而，这并不意味着高斯过程回归没有**缺点**。对于大规模的数据集，高斯过程回归可能会变得非常计算密集，并且可能需要大量的内存。此外，选择和调整合适的核函数可能会变得比较复杂。

适用场景

机器学习和数据科学：在预测复杂的、非线性的数据关系时，GPR可以被用作一个强大的工具。它可以处理回归问题，也可以被用于分类问题。例如，它可以被用来预测股票价格，或者预测疾病的发展。

优化问题：GPR常被用在贝叶斯优化（Bayesian Optimization）中。贝叶斯优化是一种序列决策策略，用于在黑盒函数优化中寻找最优解。它在计算复杂、观察成本高昂、无法直接应用梯度信息的情况下，特别有用。例如，用于调整机器学习模型的超参数、优化实验设计、或者在强化学习中寻找最佳策略。

空间统计和地理信息系统（GIS）：在空间统计和GIS中，GPR可以被用来进行空间插值（即预测地理空间中的数据点），如预测气候变化或者土壤成分。

时间序列分析：在金融、经济、气候研究等领域，GPR可以用于处理和预测时间序列数据。

机器人技术和控制系统：在控制系统和机器人技术中，GPR可以用于学习和预测系统的动态。例如，预测机器人的移动，或者预测飞行器的轨迹。

这次项目的公司股票预测，就属于时间序列分析的范畴。

2.论文分析与GPR数学原理介绍

我首先通读了所给的11篇参考文献，了解了高斯过程回归的原理与应用，最终在预测股票时选择采用第6篇“Long-term Stock Market Forecasting using Gaussian Processes”论文中的模型，我将主要介绍该片论文并说明我用到其他论文中的内容，具体如下：

2.1.股票市场介绍

股票市场是以约定价格交易公司股票的公开市场。投资者（公司或个人）可以买卖股票，这笔交易被称为交易。股票价格取决于需求和供应；当有高需求时上升，而在低需求时下降。在股票市场中，季度（Q）指的是一年中的四分之一。四个季度分别是：1月、2月和3月（Q1）；4月、5月和6月（Q2）；7月、8月和9月（Q3）；以及10月、11月和12月72日（Q4）。投资者利用过去几个季度来预测股票的未来走势。股票市场被认为是各国的经济指标之一，股票价格的增长吸引了投资者，并增加了公司的价值。总的来说，股票市场的增长反映了各国经济的实力和发展，因此各国会关注并干预股票市场。

预测股票市场价格对来自不同领域的研究人员来说都极具吸引力。一般来说，预测股价有两种方法：基本面分析和技术分析。基本面分析依赖于每家公司过去的业绩来做出预测；技术分析则通过处理过去的股票价格，了解其模式变化并预测未来的价格。大多数机器学习应用程序对技术分析表现出更多的兴趣。

该预测的准确性对于市场经销商来说至关重要。如今，大多数股票市场交易员依靠机器学习技术来分析和预测股价和指数的变化。由于季节、政治局势和经济条件等因素导致股市波动，这些技术的准确性仍然不令人满意。这种波动并不总是遵循确切的季节周期，但在预测时我们不仍应忽略这种波动。这点在我进行预测时也需要考虑。

2.2.高斯过程(GP)

高斯过程（GP）是机器学习中的一种流行技术，广泛用于时间序列分析。Rasmussen和Williams将GP定义为"随机变量的集合，其中任何有限个都具有联合高斯分布"。GP用于通过定义两个函数来表征函数上的概率分布：均值函数$m(x)$和协方差函数均值函数$k(x_1, x_2)$为了将一个真实的过程f(x)描述为一个GP。表示为

$$f(x) \sim GP(m(x), k(x_1, x_2))$$

其中,

$$m(x) = E[f(x)]$$
$$k(x_1, x_2) = E[(f(x_1) - m(x_1))(f(x_2) - m(x_2))]$$

在回归中，对给定的N维数据集D：$D = \{(x_i, y_i)|i = 1, \ldots, N\}$，其中$x_i \in \mathbb{R}^D$且$y_i \in \mathbb{R}$, 从而在给定$x_*$的情况下使用f(x)预测新的$y_*$，例如用$f(x)$ 预测，$y_i = f(x_i) + \delta_i$其中δ_i是均值为零且方差为σ^2的高斯噪声。

由第五篇参考文献，我们知道股市收盘价格是无噪声的，因为真实价格是在收盘时评估的，观测目标y的先验分布由下式给出

$$y \sim N(0, K(X, X))$$

其中，$K(X, X)$是所有训练点对之间的协方差矩阵，X是输入的（n×m）矩阵。在这个项目中，论文中研究人员使用了RBF核，表达式如下：

$$k(x_1, x_2) = \exp(-\sigma||x_1 - x_2||^2)$$

当然，除了此论文中直接使用的RBF，我们可以选择不同的核函数，这部分在文献五中较为详细，我将在下一部分介绍。

在此之后，可以通过以训练数据为条件来计算，得到条件概率$p(f(x_*)|x_*, D)$，从而得到y_*的预测分布。y上的联合分布和x*的预测如下：

$$\begin{bmatrix} y \\ f(x_*) \end{bmatrix} \sim N\left(0, \begin{bmatrix} K(X, X) & K(X, x_*) \\ K(x_*, X) & K(x_*, x_*) \end{bmatrix}\right)$$

由y的先验分布$y \sim N(0, K(X, X))$ 便可得到y_* 的预测分布的均值和协方差如下：

$$\bar{f}(x_*) = K(X, x_*)^T (K + \sigma_n^2 I)^{-1} y,$$
$$V_f(x_*) = K(x_*, x_*) - K(X, x_*)^T (K + \sigma_n^2 I)^{-1} K(X, x_*)$$

2.3.核函数中协方差的选择与适用场景

由第五篇文献，我们也可以选择其他核函数，不同的核函数适用场景不同，现列举如下：

Squared Exponential Covariance（平方指数协方差）：也称为高斯核函数或径向基函数（Radial Basis Function，RBF），表达式如下：

$$k(t, t') = \sigma_f^2 \exp(-\frac{(t - t')}{2\rho^2})$$

其中ρ是协方差函数的特征长度尺度，σ_f控制曲线的"平滑度"。平方指数协方差是无限可微的，这意味着存在一些关于估计函数的平滑程度的隐含假设，也就是说，高阶噪声将被该函数平滑。

Matérn Class Covariance（马特尔协方差）：马特尔核函数是一族核函数，根据不同的参数值可以得到不同程度的光滑性。其表示如下：

$$k(t, t') = \frac{2^{1-\nu}}{\Gamma(\nu)} (\frac{\sqrt{2\nu}(t - t')}{\rho})^{\nu}$$

该协方差函数是平方指数协方差（SE）函数的替代方案。虽然它可以对高阶项进行建模，但它不是无限可微的，并且将SE函数近似为 $\nu \to \infty$ 。在我们的实现中，我们只使用这个协方差函数的一个特定情况，其中最常用的是Matérn 3/2 核函数。Matérn 3/2 核函数的表达式如下：

$$k_{v=\frac{3}{2}}(t, t') = (1 + \frac{\sqrt{3}(t - t')}{\ell}) \exp(-\frac{\sqrt{3}(t - t')}{\ell})$$

在大多数机器学习应用中，$k_{v=\frac{3}{2}}$ 用途更广泛，因为v的值不太高。当v过高时，预测函数将具有过多的锯齿状边缘；另一方面，当v太低时，函数将过于平滑。无论哪种情况，都对 $\hat{f}$ 做出了不切实际的假设，因此，$k_{v=\frac{3}{2}}$ 是一个很好的核函数。

Rational Quadratic Covariance（有理二次协方差）：表达式如下

$$k_{RQ}(t - t') = (1 + \frac{(t - t')^2}{2\alpha\ell^2})^{-\alpha}$$

k_{RQ}协方差是SE协方差函数中$\alpha \to \infty$ 的极限情况，被视为具有不同长度尺度的SE协方差函数的无限和。

Covariance	适用场景
Squared Exponential Covariance	广泛应用于高斯过程回归，机器学习，神经网络等领域。它允许平滑，非线性的函数映射，这是很多实际问题的关键要素。例如，在预测气候模型，股市，或者复杂系统的行为时，这种核函数非常有用。
Matérn Covariance	在统计学，尤其是空间统计和机器学习中非常有用。这种核函数提供了一种插值方法，它能够在保持一定程度的平滑性的同时，更好地捕捉数据中的高频信息。例如，如果你正在尝试预测地形数据（例如，海拔，湖泊深度等），或者在环境科学中预测空气质量或水质等参数，马特尔核函数可能是一个很好的选择。

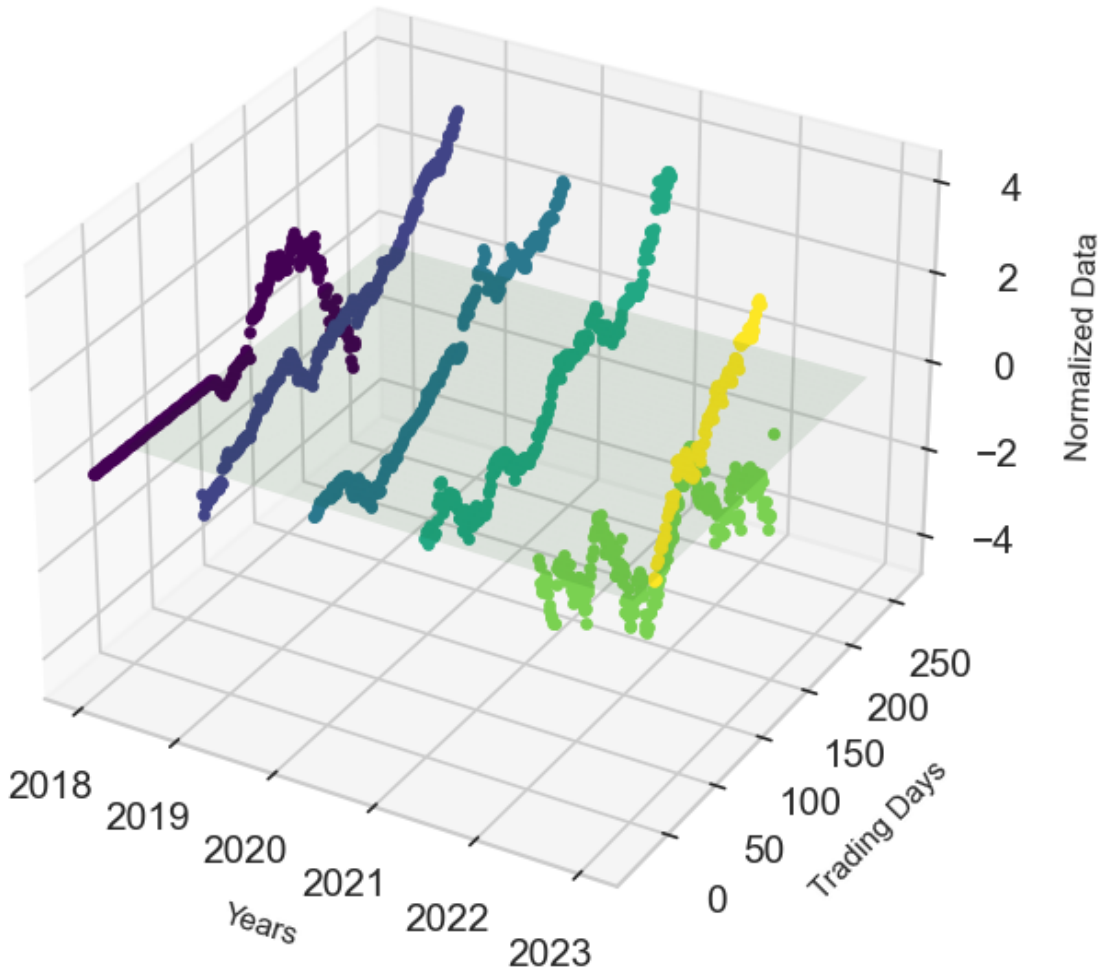
Covariance	适用场景
Rational Quadratic Covariance	常用于模拟或预测具有多尺度或长距离相互作用的问题，如在天文学中预测星系的分布或在地理信息系统中模拟地形。有理二次核函数能够捕捉到在不同尺度上的特性，这使得它在处理此类问题时效果更好。

有了不同的核函数之后，我们如何来选择呢？文献五中也给出了解答。我们来计算数据的最优超参数集即可。

2.4.方法论——如何利用高斯过程回归预测股票

根据文献6，利用高斯过程回归预测股票的主要思想是避免将整个历史表示为一个时间序列。在回归模型中，每个时间序列应被视为独立的输入变量。对于交易年份i，有M_i个交易日，$i = 1, \ldots, N$和$t = 1, \ldots, M$。高斯过程回归模型基于从$i = 1, \ldots, N-1$个交易年和从第N个交易年开始的部分交易日，$\{y_t^N\}$，$t = 1, \ldots, M$，我们想依次来预测该年余下的交易日的股票信息，即$\{y_\tau^N\}$，$\tau = M_N + 1, \ldots, M_N + H$，其中$M_N + H$是第N个交易年中最后一个交日。

我将我预测的苹果(APPL)公司的数据按此要求可视化如下：



2.5.模型参数的优化

机器学习算法的使用经常涉及学习参数和模型超参数的仔细调整。不幸的是，这种调整通常是很繁琐的，需要专家经验、经验法则，有时甚至是残酷的搜索。因此，对于能够优化任何给定学习算法对当前问题的性能的自动方法具有很大的吸引力。

通过阅读论文“Practical Bayesian Optimization of Machine Learning Algorithms”，对于高斯过程回归中核函数的参数，我决定用贝叶斯优化对高斯过程回归模型进行超参数调优。定义了一个高斯过程回归模型，并指定了搜索超参数的范围。然后，定义了一个优化目标函数，该函数将高斯过程回归模型的超参数设置为给定的参数，并返回交叉验证得到的负平均绝对误差（我们希望最小化这个值，所以取负数）。然后，这个函数使用BayesSearchCV进行贝叶斯优化，寻找最优的超参数。

机器学习里面的超参数优化，即使用的模型为 f ，超参数为输入的 x ，评估指标为输出的目标函数值，在这个场景下，很多机器学习的入门课程都会提到网格搜索和随机搜索，但是这两个其实本质上也是一种类似于穷举的方式，随便选取一组可能的x，然后分别计算目标值，最后对比所有的结果得到最好的解，可以看出来这种求解是很低效的，因此，解决这种问题需要设计一种高效的算法，来在有限的时间里面找到一个相对不错的解，这就是贝叶斯优化。

贝叶斯优化，是一种使用贝叶斯定理来指导搜索以找到目标函数的最小值或最大值的方法，就是在每次迭代的时候，利用之前观测到的历史信息（先验知识）来进行下一次优化，通俗点讲，就是在进行一次迭代的时候，先回顾下之前的迭代结果，结果太差的x 附近就不去找了，尽量往结果好一点的x附近去找最优解，这样一来搜索的效率就大大提高了，这其实和人的思维方式也有点像，每次在学习中试错，并且在下次的时候根据这些经验来找到最优的策略。

贝叶斯优化（Bayesian Optimization），主要用来解决计算成本昂贵的黑盒优化问题，这种问题有着以下两个特点：

1.目标函数 $f(x)$ 及其导数未知，否则就可以用梯度下降等方法求解

2.计算目标函数时间成本大，意味着像蚁群算法、遗传算法这种方法也失效了，因为计算一次要花费很多时间

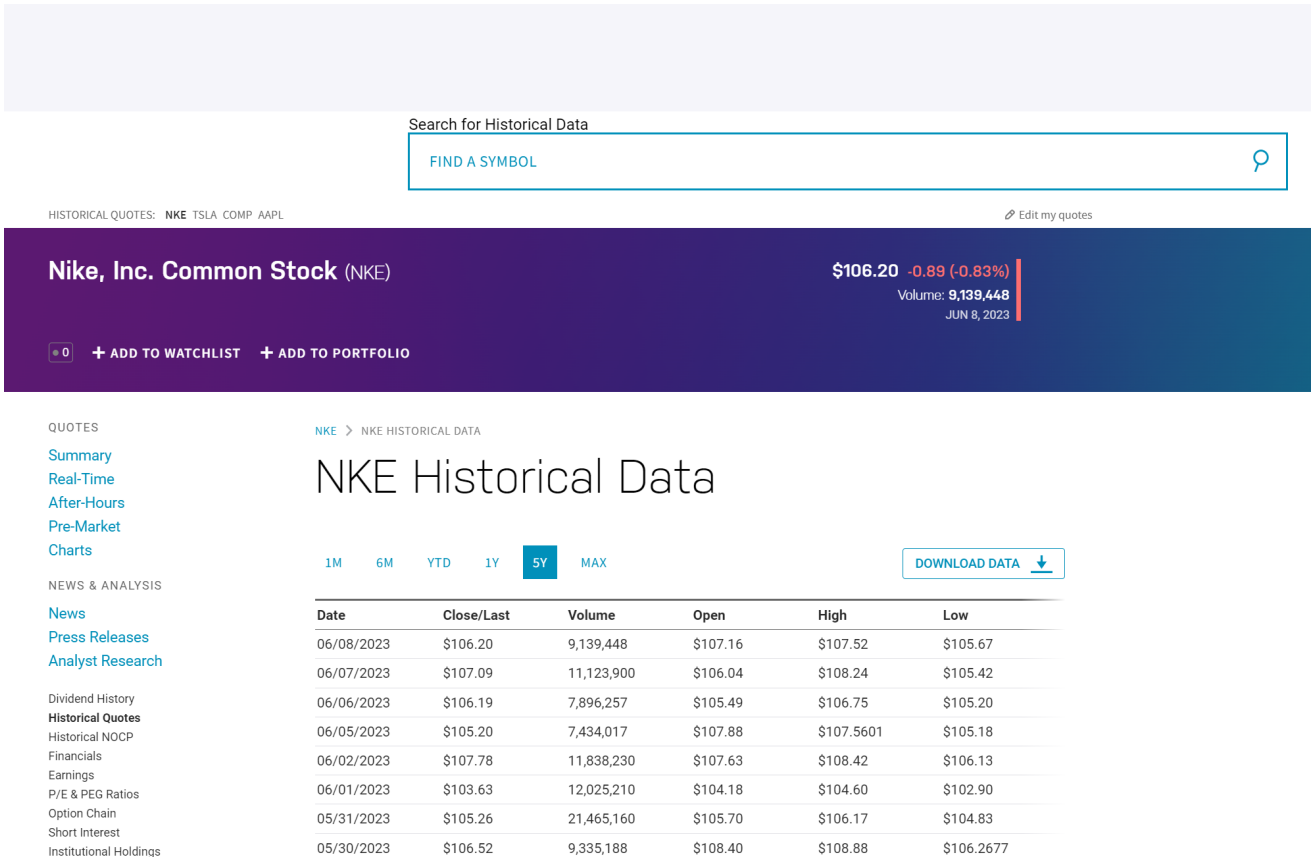
我使用贝叶斯优化的框架来考虑这个问题，学习算法的泛化性能被建模为来自高斯过程（GP）的样本。对GP性质的某些选择，如内核的类型及其超参数的处理，可以在获得一个可以实现专家级性能的良好优化器方面发挥关键作用。

具体实现在3.3部分说明

3.GPR预测股票

3.1.数据描述与预处理

使用了nasdaq官网上导出的5个公司（APPL、HPE、NKE、SBUX、TSLA）的数据集。其中为了和文献6对比，下载了SBUX与HPE，然而，文献6中的PE公司已经成为HPE的前身，不再存在。我将通过SBUX的预测来对比我的模型与文献6中的模型。



我选择了每个公司5年以来的数据，导出的csv文件部分如下

Date	Close/Last	Volume	Open	High	Low
06/06/2023	$179.21	64848370	$179.97	$180.12	$177.43
06/05/2023	$179.58	121946500	$182.63	$184.95	$178.04
06/02/2023	$180.95	61996910	$181.03	$181.78	$179.26
06/01/2023	$180.09	68901810	$177.70	$180.12	$176.93
05/31/2023	$177.25	99625290	$177.33	$179.35	$176.76
05/30/2023	$177.30	55964400	$176.96	$178.99	$176.57
05/26/2023	$175.43	54834980	$173.32	$175.77	$173.11
05/25/2023	$172.99	56058260	$172.41	$173.90	$171.69
05/24/2023	$171.84	45143490	$171.09	$172.42	$170.52
05/23/2023	$171.56	50747260	$173.13	$173.38	$171.28
05/22/2023	$174.20	43570930	$173.98	$174.71	$173.45

由文献5，股市收盘价格是无噪声的，另外，选择使用调整后的收盘价是因为我们的目标是预测股票的走势，而不是价格。调整后的收盘价用于避免股息和分割的影响，因为当股票进行分割时，其价格会下跌一半，因此我采用表格前两列的数据，即Date与Close/Last。

需要注意的是读入数据时需要去掉Close/Close列的'$'符号并转为浮点数类型。

特别需要注意的是，下载的csv文件中时间是由近期到远期，为了之后数据处理的方便，我需要将读入的数据逆序。

```
    def __load_data(self, csv_name: str):
        self.df = pd.read_csv('Data/' + csv_name + '.csv')
        self.df = self.df.iloc[:, [0, 1]]
        for i in range(len(self.df["Close/Last"])):
            self.df["Close/Last"][i] = float(self.df["Close/Last"][i][1:]) # 去掉'$'并转为浮点数
        self.df = self.df.dropna()
        self.df.Date = pd.to_datetime(self.df.Date)          # 将日期转化为Date类
        self.df = self.df.iloc[::-1].reset_index(drop=True) # 逆序
        self.quarters = ['Q' + str(i) for i in range(1, 5)]
```

预处理

1.调整后的收盘价格标准化为零均值和单位标准差，另外，还将每年的价格标准化，以避免与前几年相比的变化（公司整体市值的变动），即用每年的所有交易日减去第一个交易日标准化后的数据。

```
    def __add_normalized_data(self, df):
        normalized = pd.DataFrame()

        self.years = list(df.Date)
        self.years = list({self.years[i].year for i in range(0, len(self.years))})

        for i in range(0, len(self.years)):
            prices = self.get_year_data(year=self.years[i], normalized=False)
            mean = np.mean(prices)
            std = np.std(prices)
            prices = [(prices[i] - mean) / std for i in range(0, len(prices))]    # 单位化
            prices = [(prices[i] - prices[0]) for i in range(0, len(prices))]     # 标准化
            normalized = normalized._append(prices, ignore_index=True)

        return normalized
```

2.由于股票市场分为交易日与非交易日，每年的交易日数量不同，为方便预测我把每个交易年的交易日定为252天，文献6中为250天，由于我将每年分为4个季度，为保证每个季度所含天数相同，我选择取了4的倍数252天为交易日数量。

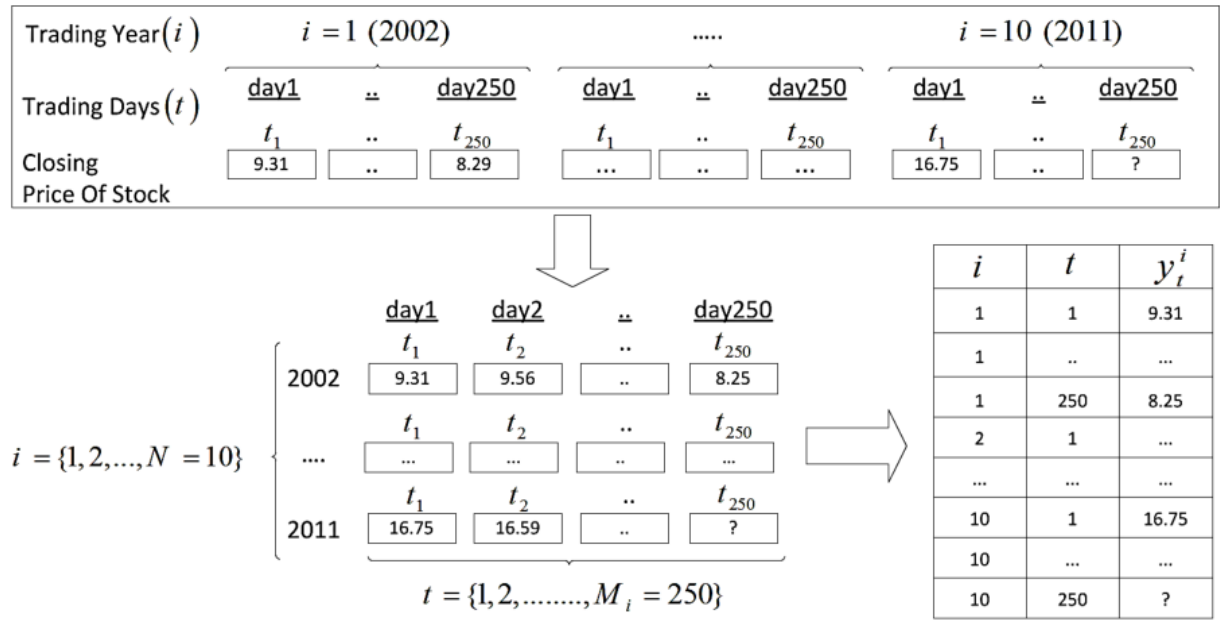
在取定长的数据时，我将超过252天的年份中超出的部分忽略，不足252天的年份中补充数据为股票的均值以保证均值不变

需要注意的是，在预处理步骤1中单位化后均值为0，而标准化后均值不为0，对天数不足252天的数据进行插值时，并非单纯的补0

```
    def get_equal_length_prices(self, normalized=True):
        df = self.__shift_first_year_prices()

        for i in range(1, len(self.years)):
            df = pd.concat([df, pd.DataFrame(self.get_year_data(year=self.years[i], normalized=normalized))], axis=1)

        df = df[:self.max_days]     # 取252天的数据


        quarters = []
```

```
        for j in range(0, len(self.quarters)):
            for i in range(0, self.max_days // 4):
                quarters.append(self.quarters[j])
        quarters = pd.DataFrame(quarters)                    # 数据按季度划分

        df = pd.concat([df, quarters], axis=1)
        df.columns = self.years + ['Quarter']
        df.index.name = 'Day'

        self.__fill_last_rows(df)

        return df
```

3.由2.5方法论部分，我们不应把全体数据作为时间序列，而应划分为交易年和该年的交易日，将每个交易年作为时间序列。

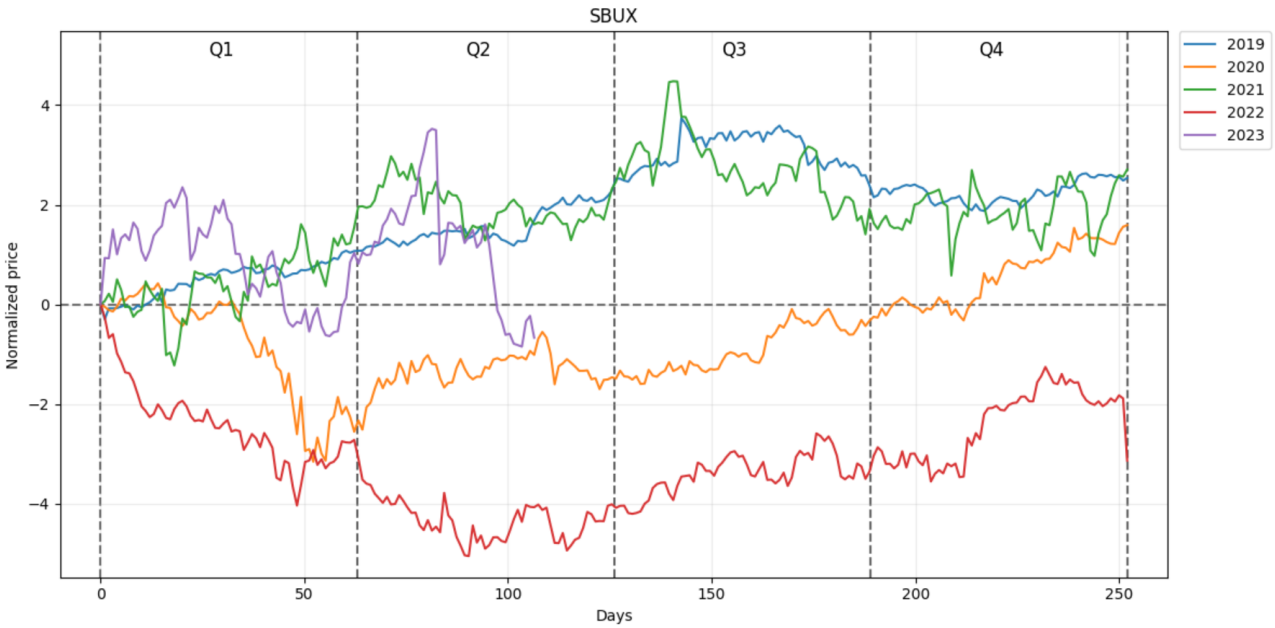


这点在读文件时已实现，如下代码是获取每一年的数据。

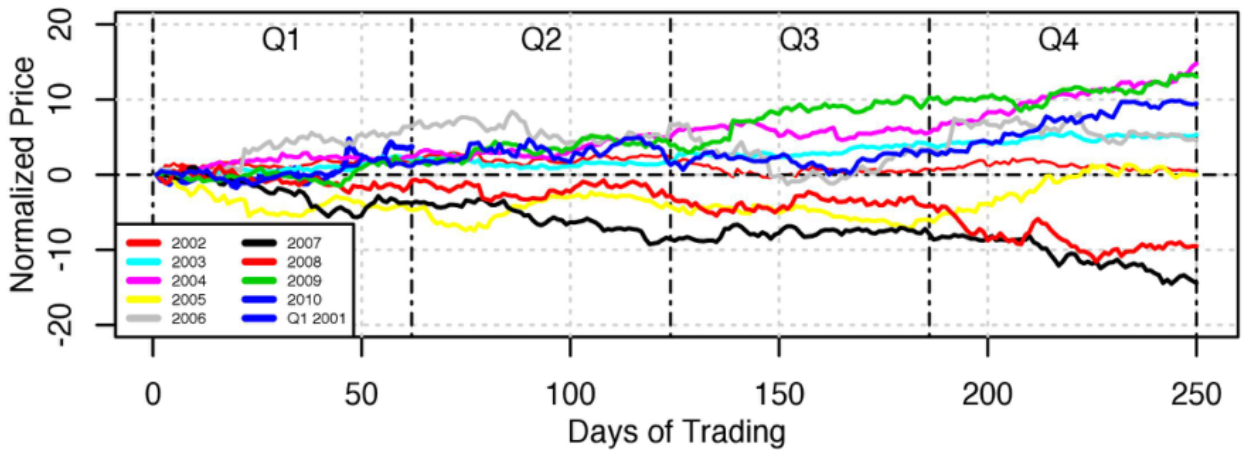
```
    def get_year_data(self, year: int, normalized=True):
        if year not in self.years:
            raise ValueError('\n' +
                             'Input year: {} not in available years: {}'.format(year,
self.years))

        prices = (self.df.loc[self.df['Date'].dt.year == year])
        if normalized:
            return np.asarray(prices.loc[:, 'Norm Close/Last'])
        else:
            return np.asarray(prices.loc[:, 'Close/Last'])
```

最终预处理结束后得到的数据可视化如下，为方便与文献6对比，我同样选择星巴克公司的数据。

我的数据预处理结果：



文献6中预处理结果



在我看来，数据预处理部分很好的复现了该论文的思路。

3.2.贝叶斯优化寻找最优参数

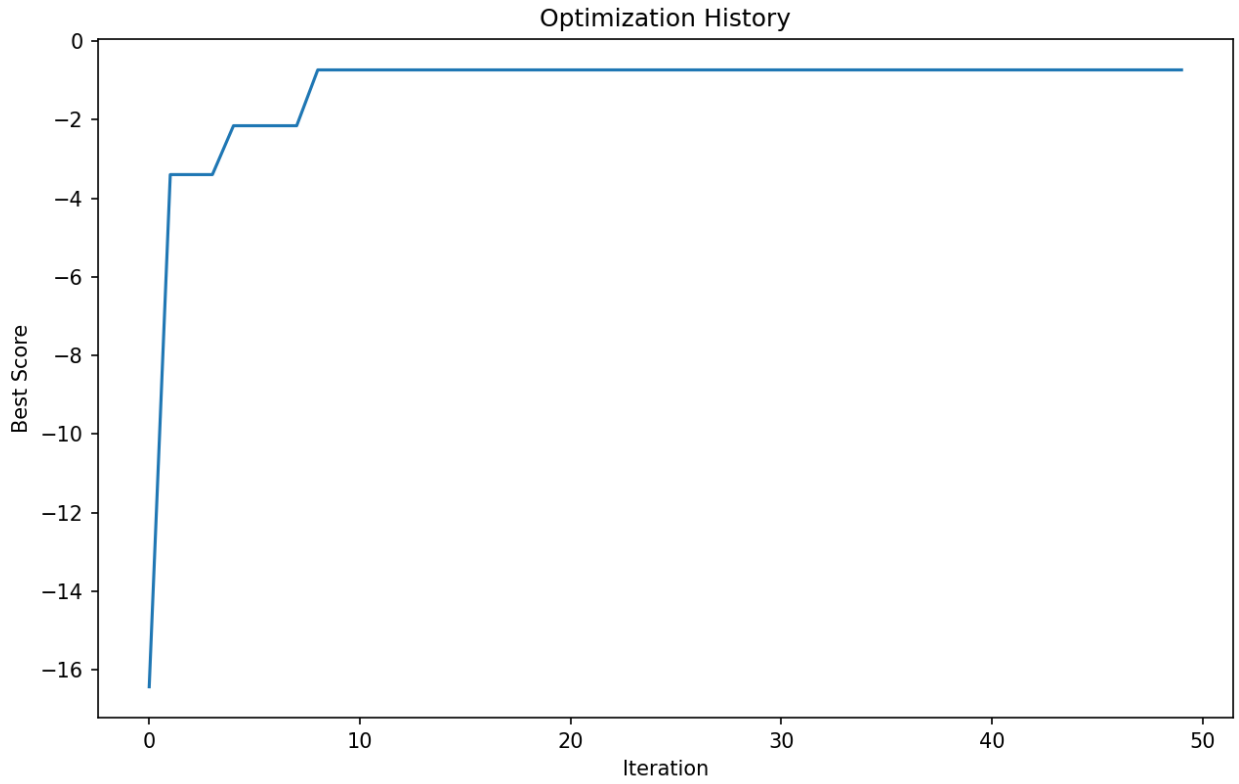
该部分我采用贝叶斯优化寻找了核函数为RBF()时的三个参数的最优

按照论文“Practical Bayesian Optimization of Machine Learning Algorithms”中思想，贝叶斯优化，主要用于寻找高斯过程回归模型的最优参数。它使用了BayesSearchCV方法，这是一个使用贝叶斯优化的交叉验证搜索，以找到最优化的模型参数。

最终，我将每次调优的结果可视化

```
    def bayesian_optimization(self):
        X, Y = self.get_eval_model()
        kernel = C() * RBF()
        search_spaces = {
            'alpha': Real(1e-10, 1e-1, prior='log-uniform'),
            'kernel__k2__length_scale': Real(1e-1, 1e1, prior='log-uniform'),
            'kernel__k1__constant_value': Real(1.0, 1e2, prior='log-uniform')
        }
```

```
history = []  # 存储优化历史

# 定义回调函数
def on_step(optim_result):
    # 获取当前的最优分数
    score = -optim_result.fun
    history.append(score)
    print("Best score: %s" % score)

 # 实例化BayesSearchCV
opt = BayesSearchCV(
    GaussianProcessRegressor(kernel=kernel, normalize_y=False),
    search_spaces,
    n_iter=3,  # 迭代次数
    cv=5,  # 交叉验证折数
    n_jobs=-1  # 使用的CPU核数
)

# 开始优化
with parallel_backend('threading', n_jobs=1):
    opt.fit(X, Y, callback=on_step)  # 传入回调函数

self.__opt = opt

# 打印最终结果
print("Val. score: %s" % opt.best_score_)
print("Test score: %s" % opt.score(X, Y))
print("Best params: %s" % str(opt.best_params_))

# 绘制优化历史
plt.figure(figsize=(10, 6))
plt.plot(history)
plt.xlabel("Iteration")
plt.ylabel("Best Score")
plt.title("Optimization History")
plt.show()
```



3.3.核函数

我使用了不同的核函数并做出对比分析

首先，我先介绍sklearn.gaussian_process库中GaussianProcessRegressor的使用方法。

`sklearn.gaussian_process`: Gaussian Processes

The `sklearn.gaussian_process` module implements Gaussian Process based regression and classification.

User guide: See the Gaussian Processes section for further details.

`gaussian_process.GaussianProcessClassifier`([...])	Gaussian process classification (GPC) based on Laplace approximation.
`gaussian_process.GaussianProcessRegressor`([...])	Gaussian process regression (GPR).

Kernels:

`gaussian_process.kernels.CompoundKernel`(kernels)	Kernel which is composed of a set of other kernels.
`gaussian_process.kernels.ConstantKernel`([...])	Constant kernel.
`gaussian_process.kernels.DotProduct`([...])	Dot-Product kernel.
`gaussian_process.kernels.ExpSineSquared`([...])	Exp-Sine-Squared kernel (aka periodic kernel).
`gaussian_process.kernels.Exponentiation`(...)	The Exponentiation kernel takes one base kernel and a scalar parameter p and combines them via
`gaussian_process.kernels.Hyperparameter`(...)	A kernel hyperparameter's specification in form of a namedtuple.
`gaussian_process.kernels.Kernel`()	Base class for all kernels.
`gaussian_process.kernels.Matern`([...])	Matern kernel.
`gaussian_process.kernels.PairwiseKernel`([...])	Wrapper for kernels in sklearn.metrics.pairwise.
`gaussian_process.kernels.Product`(k1, k2)	The `Product` kernel takes two kernels k_1 and k_2 and combines them via
`gaussian_process.kernels.RBF`([length_scale, ...])	Radial basis function kernel (aka squared-exponential kernel).
`gaussian_process.kernels.RationalQuadratic`([...])	Rational Quadratic kernel.
`gaussian_process.kernels.Sum`(k1, k2)	The `Sum` kernel takes two kernels k_1 and k_2 and combines them via
`gaussian_process.kernels.WhiteKernel`([...])	White kernel.

根据GaussianProcessRegressor官方文档 有如下参数

kernel： 内核函数，即GP的协方差函数，默认为空

alpha： 拟合期间添加到核矩阵对角线的值, 通过确保计算值形成正定矩阵，这可以防止在拟合过程中出现潜在的数值问题。它也可以解释为训练观测值上附加高斯测量噪声的方差。默认为1e-10

optimizer： 用于优化内核参数的内部支持的优化器之一，由字符串指定，也可以是作为可调用对象传递的外部定义的优化器。默认为"fmin_l_bfgs_b"方法

n_restarts_optimizer： 为找到最大化对数边际似然的内核参数而重新启动优化器的次数。优化器的第一次运行是从内核的初始参数执行的，其余参数（如果有的话）来自 thetas 从允许的 theta 值空间中随机采样对数均匀。默认为0

normalize_y： 是否通过移除均值并缩放到单位方差来标准化目标值。建议在使用零均值、单位方差先验的情况下使用。, 默认为False

copy_X_train： 如果为 True，训练数据的持久副本将存储在对象中。否则，只会存储对训练数据的引用，如果外部修改数据，这可能会导致预测发生变化。默认为True

random_state： 确定用于初始化中心的随机数生成。默认为空

另外，介绍sklearn.gaussian_process库中的核函数的选择

根据前期对不同核函数使用场景的分析，我最终使用并对比了两种核函数，分别为$Matérn_{\frac{3}{2}}$与RBF核函数

sklearn.gaussian_process.kernels.RBF与之前的介绍相同，参数为

length_scale： 内核的长度尺度

length_scale_bounds： "length_scale"的下限和上限。如果设置为"fixed"，则在超参数调整期间无法更改"length_scale"

sklearn.gaussian_process.kernels.Matern前前个参数与RBF相同，毕竟已经介绍过两者相比，Matern它有一个额外的参数ν控制结果函数的平滑度。其另一个参数

nu： 浮点数，默认值=1.5

参数 nu 控制学习函数的平滑度。nu 越小，逼近的函数越不光滑。对于 nu=inf，内核变得等效于 RBF 内核，对于 nu=0.5 内核等效于绝对指数内核。重要的中间值是 nu=1.5（一次可微函数）和 nu=2.5（两次可微函数）。

高斯过程回归封装如下

```
# 内核参数与alpha由贝叶斯优化给出
kernel, self.__alpha = bayesian_optimization.Optimization(company_name).get_best_para()
self.__iterations = 10
self.__kernels = [kernel]
self.__gp = GaussianProcessRegressor(kernel=self.__kernels[0], alpha=self.__alpha,
                                     n_restarts_optimizer=self.__iterations,
                                     normalize_y=False)
```

3.4.预测的输入

输入参数

输入参数X为与时间有关的数据，Y为预处理后的股票价格信息

在我考虑时间相关的输入X时，由两种想法，一种是只输入1-252表示每年的第i个交易日($1 \leq i \leq 252$)，另一种是输入三维向量包含年月日的信息。

我分析两者的优缺点如下

输入方法	优点	缺点
交易日的1-252编号	简单：它是一个单一的、连续的数字，可以直接输入到模型中。 具有持续性：这种格式强调了时间的连续性和顺序，这对于预测模型（特别是那些依赖于时间序列数据的模型）来说是很重要的。	缺乏季节性信息：这种格式不包含一年中的某个特定日期可能带来的季节性效应，比如季度报告发布、节假日等。 忽略了周末和非交易日：股市在周末和某些特定的非交易日是不开放的，而这种格式可能会忽略这一点。
三维向量年月日	更多的信息：这种格式包含了更丰富的时间信息，比如一年中的哪一天、哪个月、哪一年等，这有助于捕捉到可能存在的季节性效应。 考虑了非交易日：这种格式可以更准确地表示实际的交易日。	更复杂：相比于一个单一的数字，一个三维向量在处理上会更复杂一些，可能需要更多的数据预处理工作。 需要更多的计算资源：增加输入的维度可能会导致需要更多的计算资源，特别是当你使用的模型比较复杂，或者数据量比较大的时候。

最终，考虑到除了公司商业行为与季节相关，股票一般不具有季节周期性，因此，我使用二维向量作为输入，分别表示年份和交易日编号。

最终预测函数如下

```
def get_eval_model(self, start_year: int, end_year: int, pred_year: int, pred_quarters:
list = None):
    # 定义训练数据的年份和季度
    years_quarters = list(range(start_year, end_year + 1)) + ['Quarter']
    # 定义训练年份
    training_years = years_quarters[:-2]
    # 从价格数据中选取需要的年份和季度的数据
    df_prices =
self.__prices_data[self.__prices_data.columns.intersection(years_quarters)]

    # 初始化X（输入数据）和Y（标签）
    X = np.empty([1,2], dtype=int)
    Y = np.empty([1], dtype=float)

    # 处理开始年份的价格数据
    first_year_prices = df_prices[start_year]
    # 如果开始年份等于公司数据的最早年份，那么进行特殊处理
    if start_year == self.__company_data.years[0]:
        # 去除价格为0的数据，并且在序列最前面加入一个价格为0的数据
        first_year_prices = (first_year_prices[first_year_prices.iloc[:] != 0])
```

```
        first_year_prices = (pd.Series([0.0], index=
[first_year_prices.index[0]-1]))._append(first_year_prices)

    # 生成输入数据X（年份和日期）和Y（价格）
    first_year_days = list(first_year_prices.index.values)
    first_year_X = np.array([[start_year, day] for day in first_year_days])

    X = first_year_X
    Y = np.array(first_year_prices)

    # 处理剩余的训练年份数据
    for current_year in training_years[1:]:
        current_year_prices = list(df_prices.loc[:, current_year])
        current_year_X = np.array([[current_year, day] for day in possible_days])
        X = np.append(X, current_year_X, axis=0)
        Y = np.append(Y, current_year_prices)

    # 处理结束年份的价格数据
    last_year_prices = df_prices[end_year]
    last_year_prices = last_year_prices[last_year_prices.iloc[:].notnull()]

    last_year_days = list(last_year_prices.index.values)
    # 如果提供了预测季度，那么对结束年份的数据进行截取
    if pred_quarters is not None:
        length = 63 * (pred_quarters[0] - 1)
        last_year_days = last_year_days[:length]
        last_year_prices = last_year_prices[:length]
    last_year_X = np.array([[end_year, day] for day in last_year_days])

    # 更新输入数据X和Y
    X = np.append(X, last_year_X, axis=0)
    Y = np.append(Y, last_year_prices)

    # 根据预测季度生成预测的日期
    if pred_quarters is not None:
        pred_days = [day for day in range(63 * (pred_quarters[0]-1), 63 *
pred_quarters[int(len(pred_quarters) != 1)])]
    else:
        pred_days = list(range(0, self.__max_days))
    x_mesh = np.linspace(pred_days[0], pred_days[-1], 2000)
    x_pred = ([[pred_year, x_mesh[i]] for i in range(len(x_mesh))])

    # 使用训练数据拟合高斯过程回归模型
    self.__gp = self.__gp.fit(X, Y)
    self.__kernels.append(self.__gp.kernel_)

    # 对预测输入数据进行预测，生成预测的均值和协方差
    y_mean, y_cov = self.__gp.predict(x_pred, return_cov=True)

    # 返回预测的输入数据、预测的均值和预测的协方差
    return x_mesh, y_mean, y_cov
```

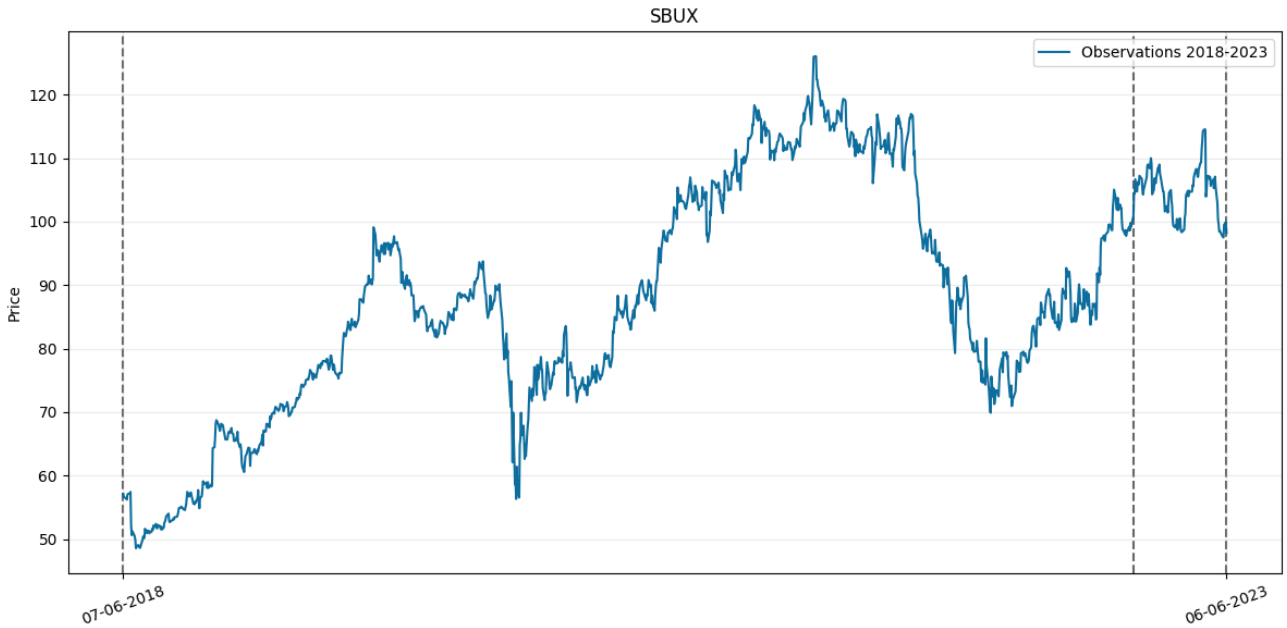
3.5.预测结果分析

数据分析前，我要首先明确一点，**我们的预测不是预测股票的精确价格，而是预测大致趋势，以期判断应该在何时买入，何时卖出，最大程度减小风险，获得最大可能的收益。**

我将对比我通过两个不同核函数得出的结果以及文献6中的结果，在此只给出我的模型预测效果三个公司的股票，其他因为篇幅原因不再列出，可以在提交的文件中查看。

星巴克（SBUX）

2018年-2023年股票收盘价总趋势



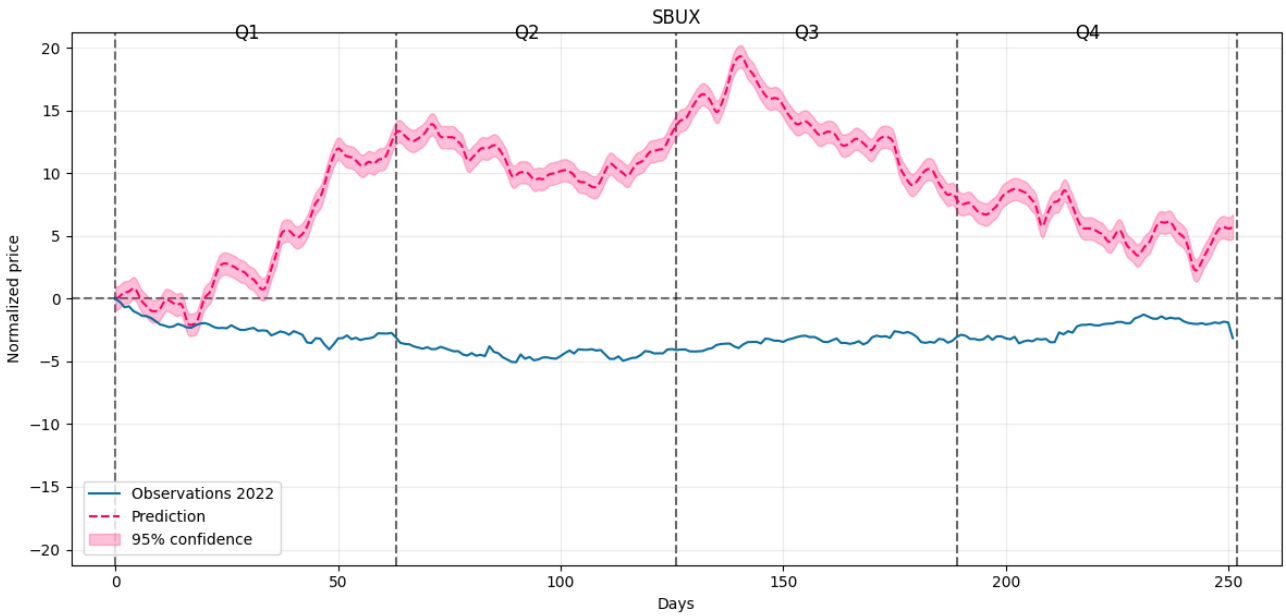
该图可以看出不同年份股价相差较大，这也是预处理时，用每年的所有交易日减去第一个交易日标准化后的数据的原因。

预处理后的数据

用2019-2021年的数据预测2022年的股票

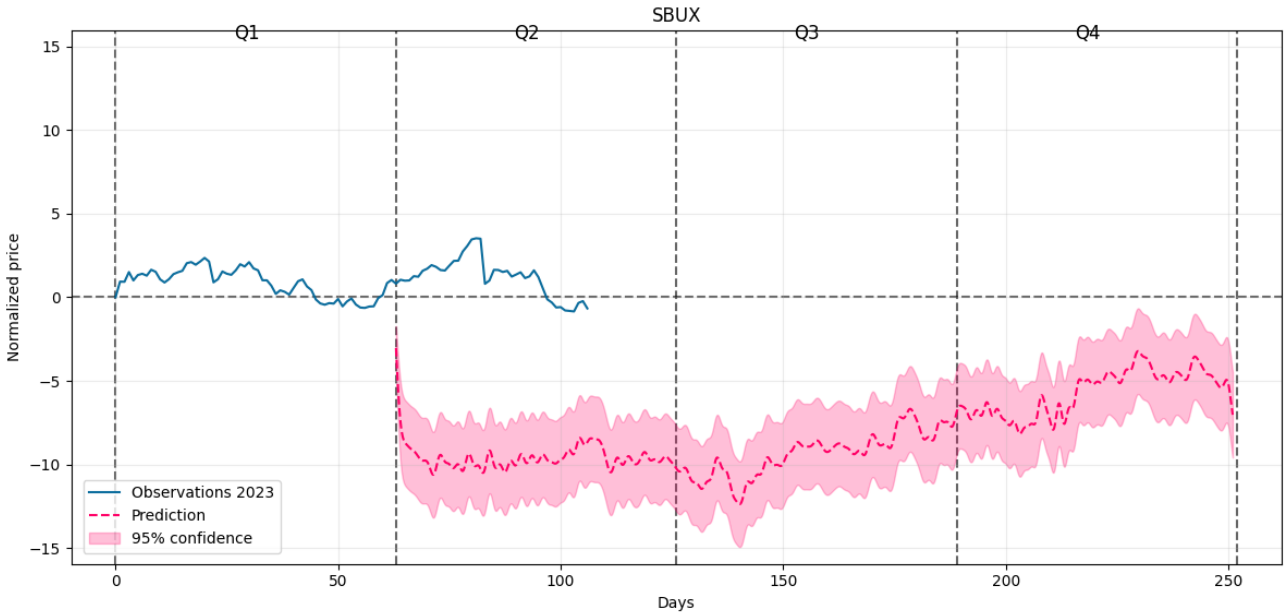
RBF核函数

$Matérn_{\frac{5}{2}}$ 核函数

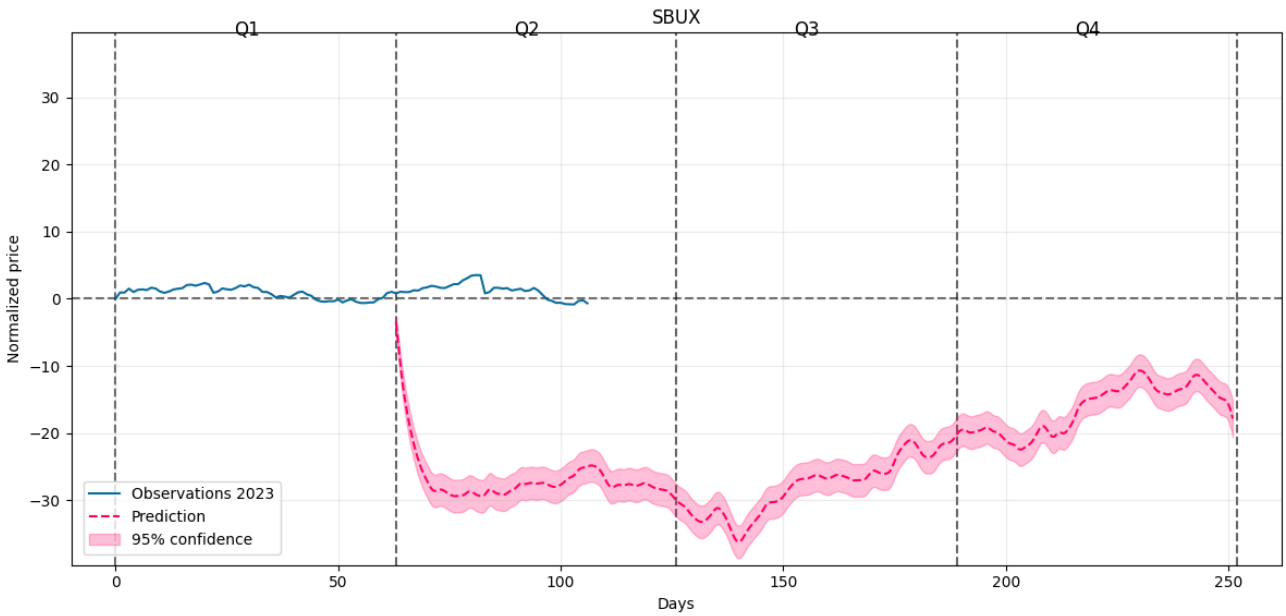


用2019-2022年与2023年第一季度的数据预测2023第二、三、四季度的股票走势

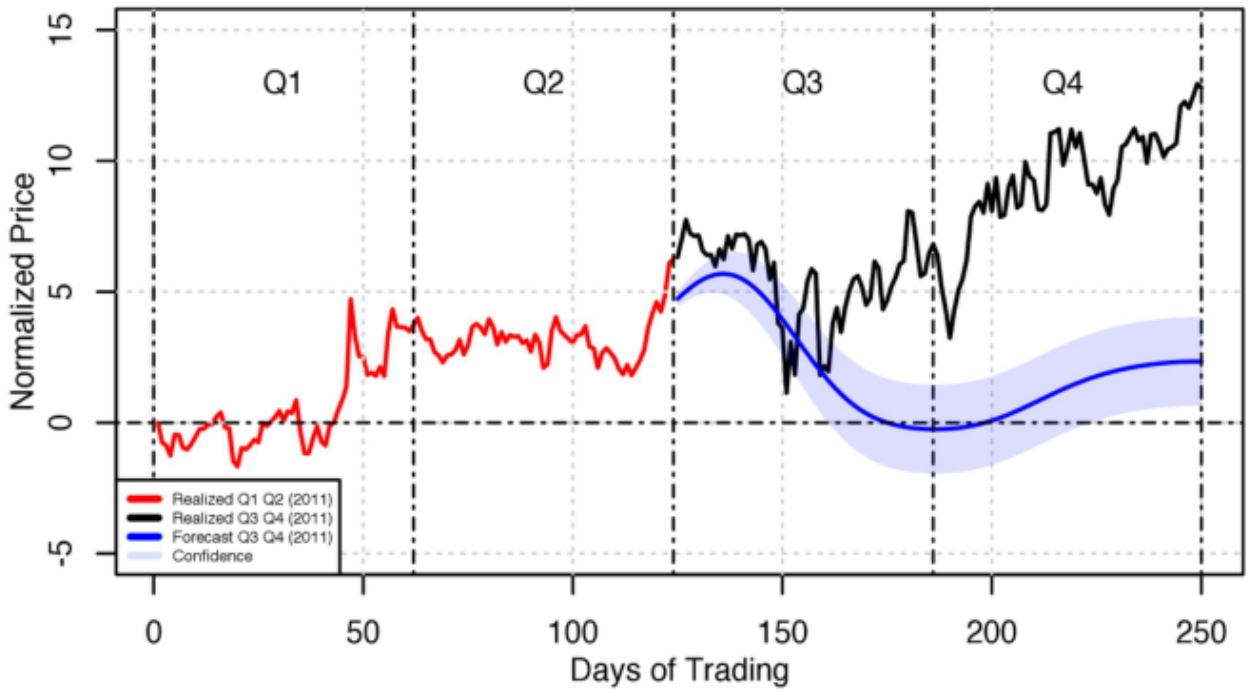
RBF核函数



$Matérn_{\frac{5}{2}}$ 核函数



文献6中预测2011年的数据



对于星巴克的股票，我们发现$Matérn_{\frac{5}{2}}$核函数的预测效果并不理想，偏差较大。此外，我们在应用该核函数对其他公司进行预测时也没有得到满意的结果，因此，我们决定不再使用该核函数。

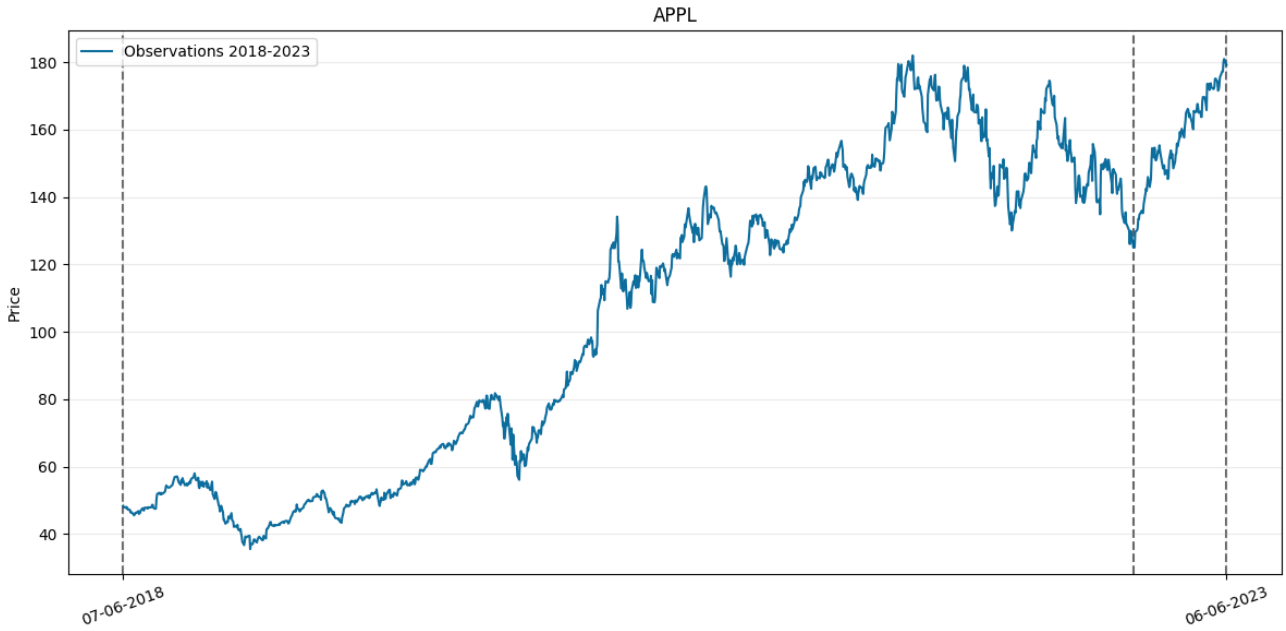
同样，RBF核函数在星巴克公司的股票预测中也表现得不尽如人意。这种预测与实际结果之间的大的偏差可能可以归因于2022年是新冠疫情影响最大的一年，这段时间内，星巴克的很多实体店都关闭了，导致股价下跌。这一结论也能从股票的总体收盘价趋势图中得到支持。

同时，我们注意到在文献6中对星巴克2011年股票价格的预测也存在较大的误差。由于我们无法精确计算误差的衡量指标，因此很难评价我所使用的模型与他们的模型的准确性和效率。但从预测曲线的走势来看，我认为我的模型在预测星巴克的股票价格上可能并不如文献6中的模型。这可能是由于外部因素的影响，导致我预测的走势与实际情况有很大的差距。

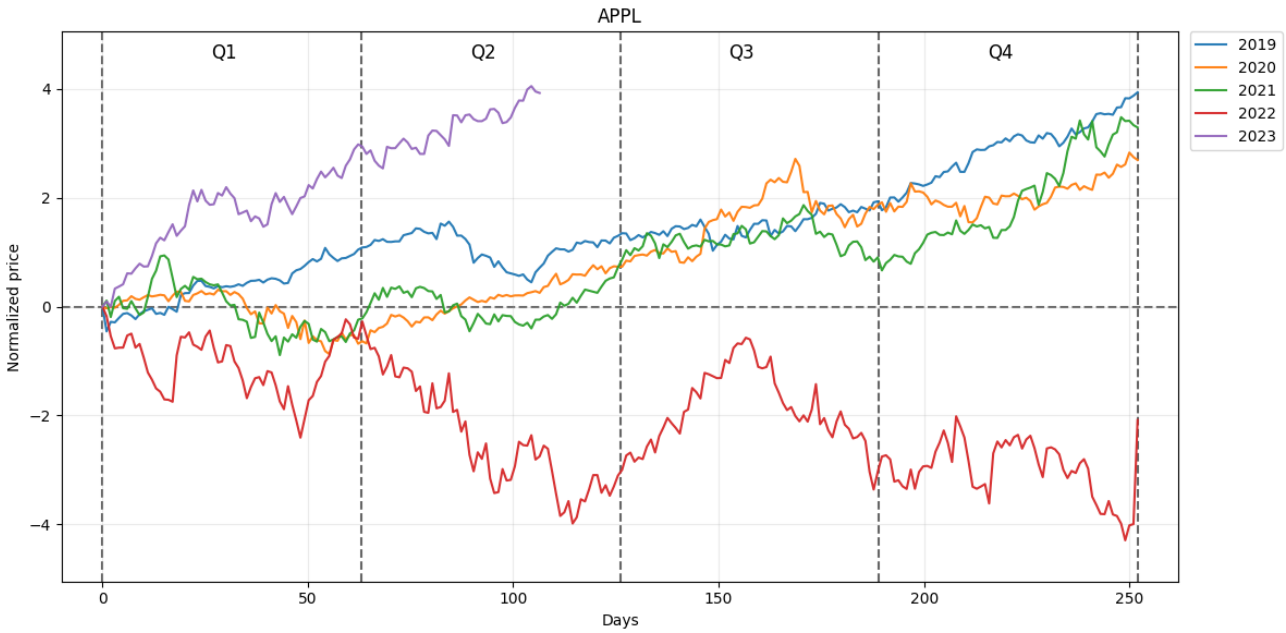
在其他两个将要介绍的公司中，我的模型预测较为可靠。

苹果（APPL）

2018年-2023年股票收盘价总趋势

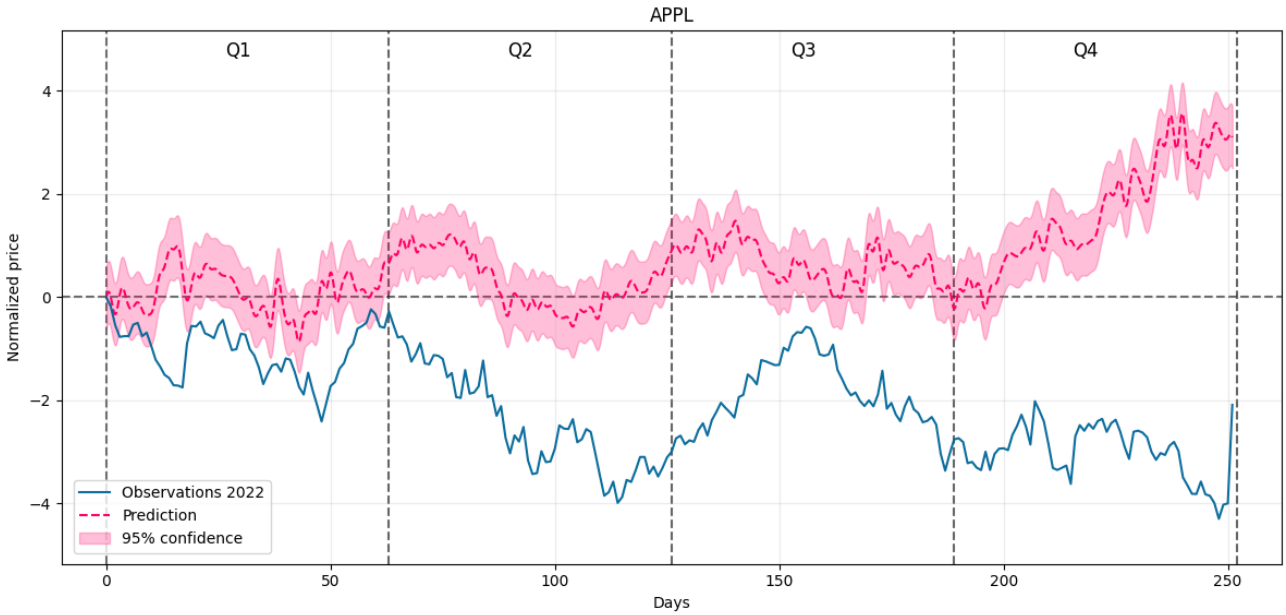


预处理后的数据



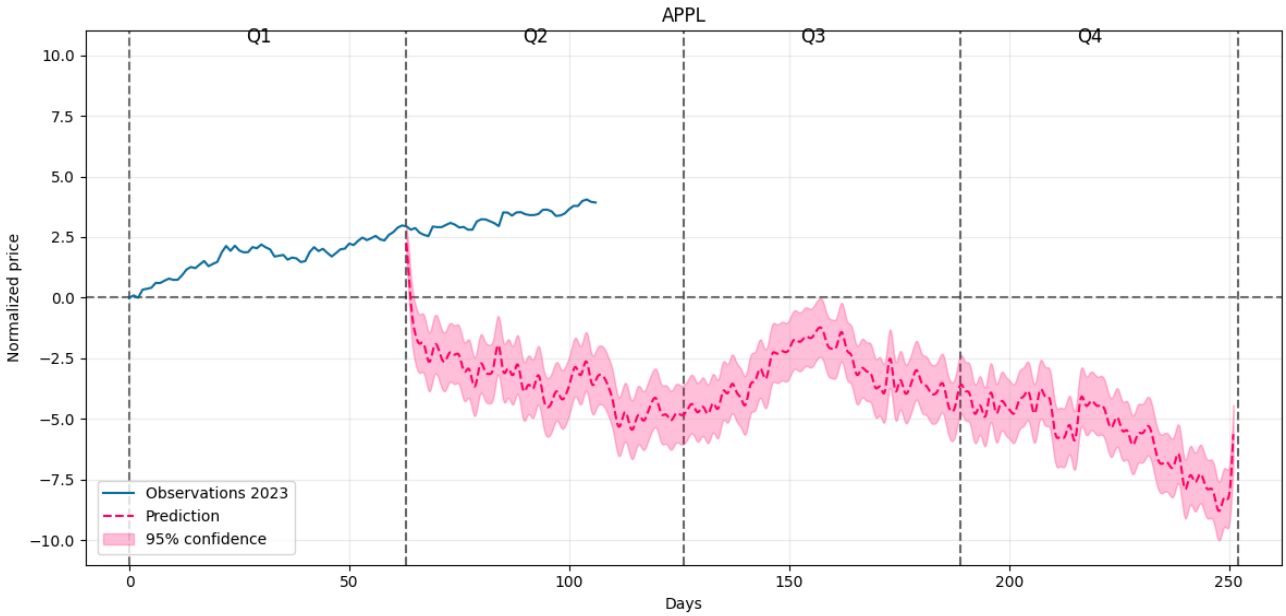
用2019-2021年的数据预测2022年的股票

RBF核函数



用2019-2022年与2023年第一季度的数据预测2023第二、三、四季度的股票走势

RBF核函数

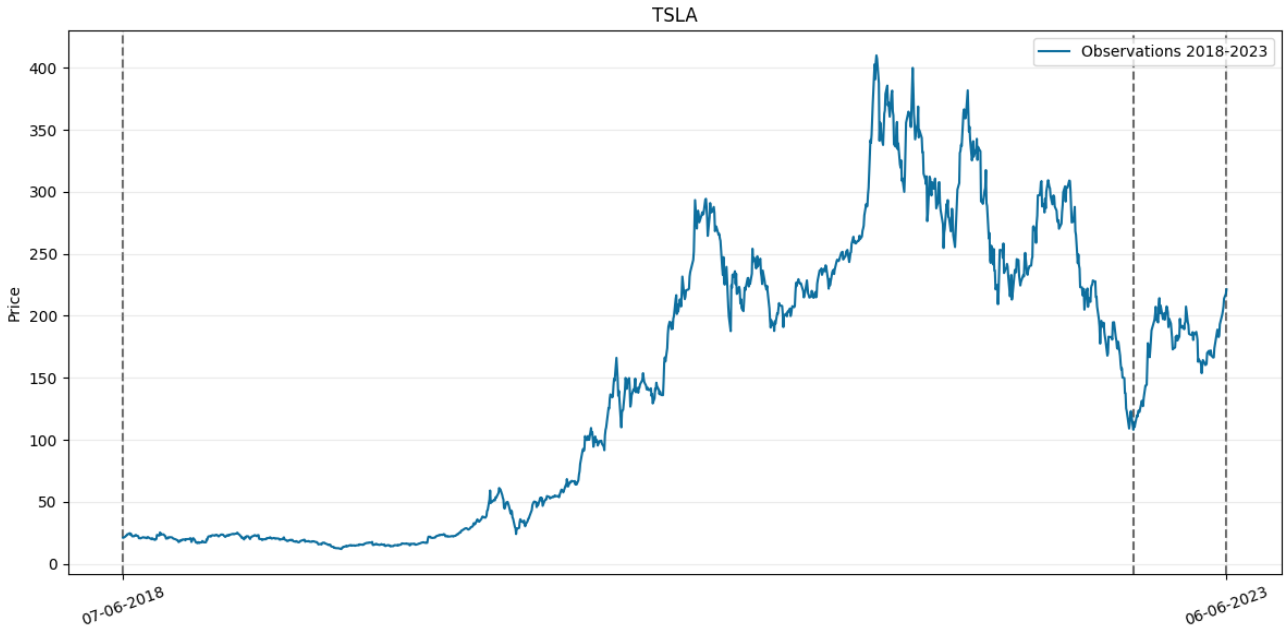


2022年苹果公司股票的价格走势能够被我们的预测模型准确捕捉。在第一季度，股价出现了频繁的震荡。随后进入第二季度，股价展现出了一个先降后升的趋势，最低点出现在大约第110个交易日。第三季度的走势呈现先升后降，但实际的最高点出现的时间比我们的预测提前。第四季度的走势则重新变得波动频繁。

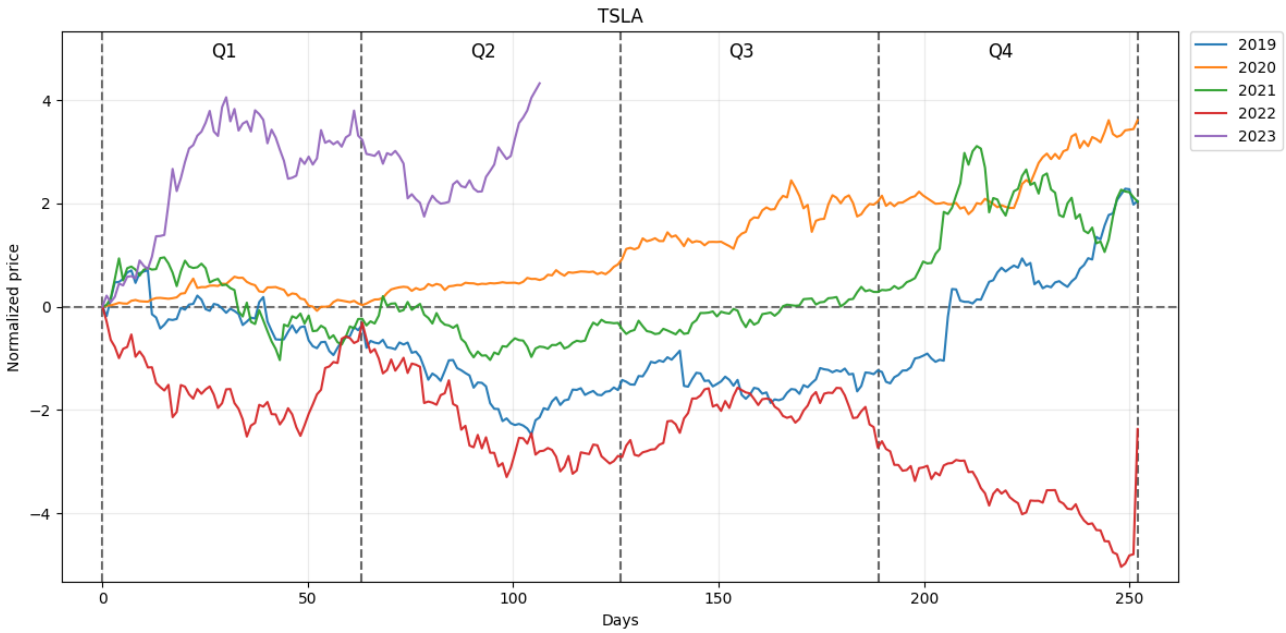
在预测2023年后三个季度时，模型开始时期低估了股市的走势。然而，从大约第70个交易日开始，模型捕捉到了一些更稳定的趋势，包括频繁的波动。根据预测，我们建议投资者在大约第160个交易日前后出售他们的股票。

特斯拉（TSLA）

2018年-2023年股票收盘价总趋势

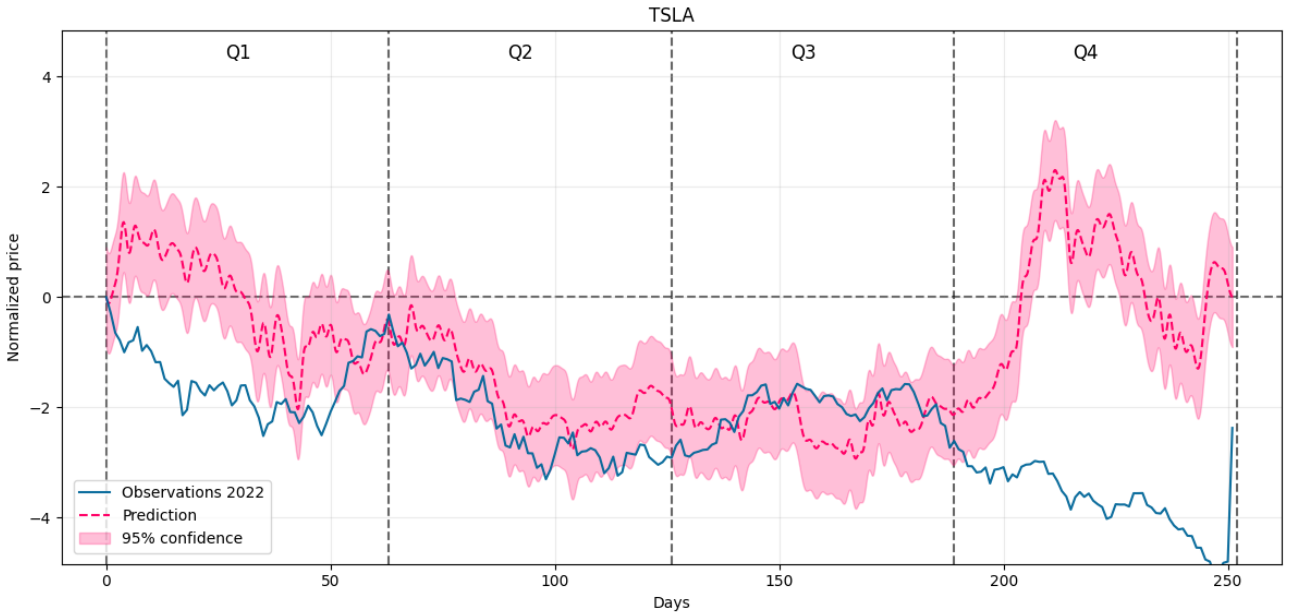


预处理后的数据



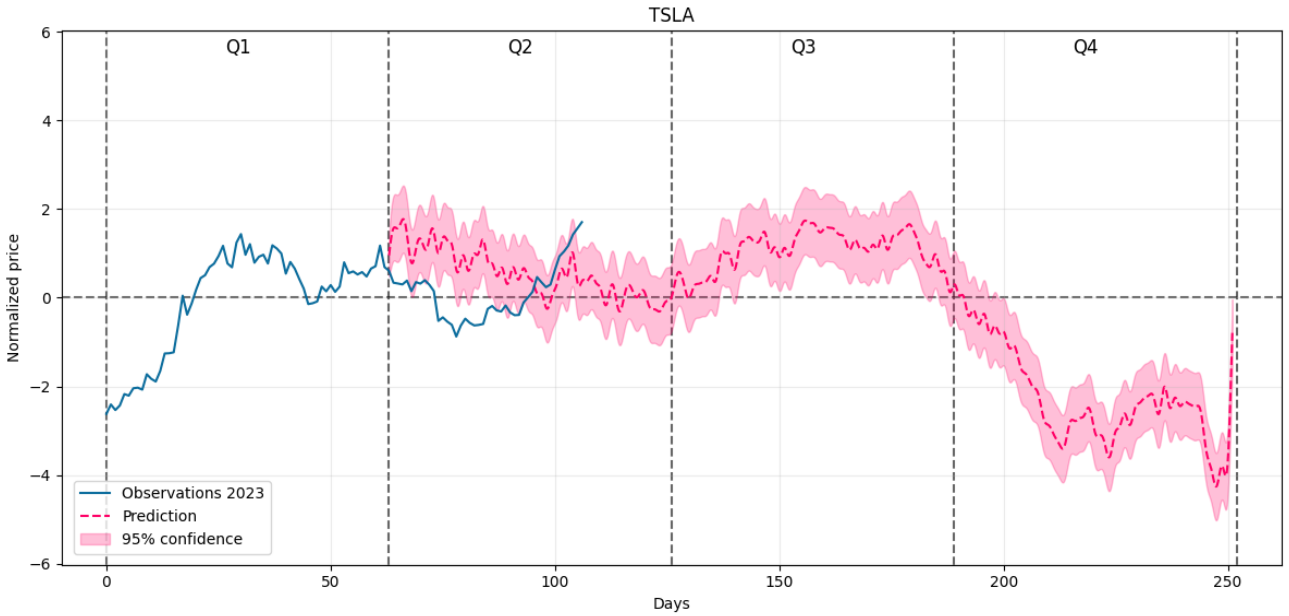
用2019-2021年的数据预测2022年的股票

RBF核函数



用2019-2022年与2023年第一季度的数据预测2023第二、三、四季度的股票走势

RBF核函数



从2022年前三个季度的特斯拉股票价格走势中找到一些明显的趋势。在第一季度，该股票的价格经历了频繁的波动。而在第二季度，我们观察到一个明显的头部反转模式，价格先是下滑至第110个交易日左右的最低点，然后反弹上升。进入第三季度，股价则表现出一种先升后降的趋势，然而，其价格峰值的出现时机提前于我们的预测。然而，第四季度的走势则与此前完全相反。

在对2023年后三个季度的预测中，我们首先看到的是一个持续的高频波动模式，这预示着市场中存在高度的不确定性。然而，在大约第190个交易日时，股价突然开始下跌，这可能预示着一个大幅度的价格调整。根据我们的预测模型，这可能是一个卖出信号，我们建议股票持有者考虑在此时点卖出他们的持仓。

****

最后，还需要指出，一般来说，这个模型只能够 “跟踪“ 股票的走势。之前已经提到，股票价格可能受到几个不可控因素的影响，例如政治局势和经济状况，这可能导致高波动，如本实验的一些领域所示。作为一个长期预测模型，不遵循这些波动是可以接受的。

4.参考资料

代码部分

我参考了github上的代码，链接如下GaussianProcesses

其中可视化部分他的代码data_plotter.py已经很完善，且考虑到本次大作业核心应该是让我们学习高斯过程，并利用其预测股票，体会随机过程的魅力，因此我直接使用了该代码。

另外，我也为复现文献6自己实现了三维的可视化部分，展现每年的标准化数据。

对于读取数据部分的代码，我在它的基础上按照我的文件进行了修改，修改部分在3.1.数据描述与预处理部分展现

对于高斯过程部分的代码，我在延用它的命名的基础的做出较大的改动，它只是使用了默认参数的核函数，我添加了自己实现并封装的贝叶斯优化class Optimization，并且采用了多种不同的核函数进行了对比。

总的来说，我在二维可视化部分完全使用了该代码，在读取文件与数据处理部分按照我的数据以及文献6的数据处理方法进行改动，高斯过程回归部分则时自己实现。

文献部分

贝叶斯优化的参考文献

“Practical Bayesian Optimization of Machine Learning Algorithms”

"Google Vizier: A Service for Black-Box Optimization"

5.一次失败的尝试

除了使用库函数之外，我也尝试自己实现高斯过程来实现短期预测，但效果很差，预测结果总会快速趋近于0

读文件部分不再赘述，未进行数据预处理。

核函数(RBF)

```
def kernel(X1, X2, l=1.0, sigma_f=1.0):
    sqdist = np.sum(X1 ** 2, 1).reshape(-1, 1) + np.sum(X2 ** 2, 1) - 2 * np.dot(X1,
X2.T)
    return sigma_f ** 2 * np.exp(-0.5 / l ** 2 * sqdist)
```

高斯过程(GP)

```
def gp_regression(X_train, Y_train, X_test, l=1.0, sigma_f=1.0, sigma_y=1e-8):
    K = kernel(X_train, X_train, l, sigma_f) + sigma_y ** 2 * np.eye(len(X_train))
    K_s = kernel(X_train, X_test, l, sigma_f)
    K_ss = kernel(X_test, X_test, l, sigma_f) + 1e-8 * np.eye(len(X_test))
    K_inv = np.linalg.inv(K)

    # 预测的均值
    mu_s = K_s.T @ K_inv @ Y_train

    # 预测的协方差
    cov_s = K_ss - K_s.T @ K_inv @ K_s

    return mu_s, cov_s
```

负似然对数调优

```
def nll_fn(X_train, Y_train, noise):
    '''
    返回一个函数，该函数计算高斯过程的负对数似然。
    '''
    def step(theta):
        K = kernel(X_train, X_train, l=np.exp(theta[0]), sigma_f=np.exp(theta[1])) + \
            noise**2 * np.eye(len(X_train))
        # 计算 Cholesky 分解（为了数值稳定性）
        L = np.linalg.cholesky(K)
        # 计算负对数似然
        return np.sum(np.log(np.diagonal(L))) + \
            0.5 * Y_train.T @ np.linalg.lstsq(L.T, np.linalg.lstsq(L, Y_train, rcond=-1)
[0], rcond=-1)[0] + \
            0.5 * len(X_train) * np.log(2 * np.pi)

    return step
```

预测

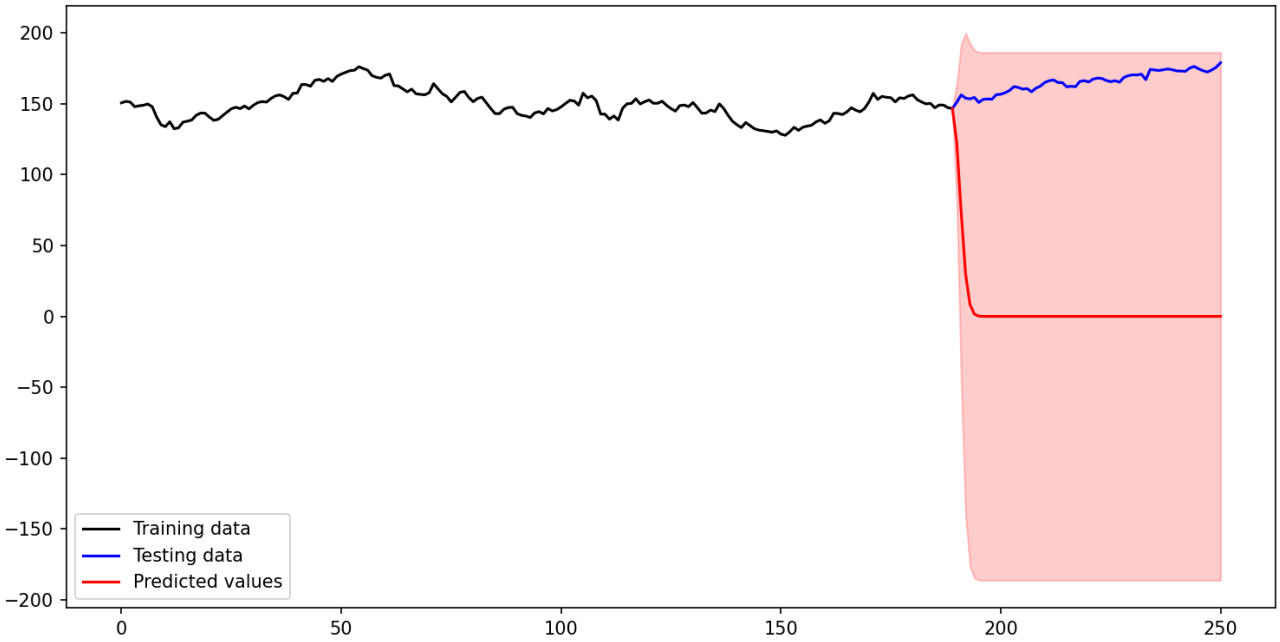
训练集、测试集按3:1划分，由于文件时间为逆序（之前已经提到），因此阅读代码时可能误以为数据集划分错误。

```
X, Y = getData()
X = X.reshape(-1, 1)
Y = Y.reshape(-1, 1)

# 训练集、测试集
test_size = int(len(X) / 4)
X_train = X[test_size - 1:]
Y_train = Y[test_size - 1:]
X_test = X[:test_size]
Y_test = Y[:test_size]


initial_guess = np.log([1, 1])

print(initial_guess)
```

```
noise = 0.3

# 使用 scipy.optimize.minimize 找到最小的负对数似然
res = sopt.minimize(nll_fn(X_train, Y_train, noise), initial_guess,
                    bounds=((1e-5, None), (1e-5, None)), method='L-BFGS-B')

# 得到最优参数
l_opt, sigma_f_opt = np.exp(res.x)

mu_s, cov_s = gp_regression(X_train, Y_train, X_test, l_opt, sigma_f_opt)
```

可视化部分也不再展示。

最终预测结果很快下降为0



这次尝试预测结果错误，我认为是调优方法不合适导致参数错误，希望从中吸取经验，在以后的项目中不断改进。

6.总结

在这个项目中，我充分体会了高斯过程的强大功能，对股票市场进行了深度和长期的预测。我选择了一些纳斯达克市场上具有代表性的股票，对其进行了高斯过程建模。这种预测方法不仅提供了对未来股价走势的预测，还能给出预测的不确定性，这在股市投资中具有重要价值。

此外，在这个项目中，我还对高斯过程回归有了更深入的理解。我学习了如何构建高斯过程模型，以及如何用它来处理实际问题。我深入探索了随机过程与随机变量的关系，理解了它们在数据分析和预测中的关键作用。尽管我的模型还存在一些不足，比如对于极端事件的处理能力有待提高，或者对于股市中一些复杂规律的捕获仍需要优化，但是总的来说，我认为这个项目给我带来了很大的收获。

这个项目的经历使我认识到，高斯过程不仅是一种强大的预测工具，更是一种理解数据和发现数据潜在规律的强大方法。在未来的工作和学习中，我将继续探索并应用高斯过程以及其他机器学习算法，不断提升自己。