基于LSTM的蔬菜类商品的自动定价与补货决策模型

摘要

蔬菜类商品的保鲜期较短，且品相随销售时间的增加而变差，大部分品种如当日未售出，隔日就无法再售。本文根据某商超近三年销售流水明细数据与蔬菜类商品的批发价格率等历史信息，基于长短期记忆网络（Long short-term memory，LSTM），建立了蔬菜类商品的自动定价与补货决策模型，并给出了未来一周的日补货总量和定价策略。

针对问题一，对蔬菜各品类和单品销售量的分布规律及相互关系进行分析。首先利用Excel对附件一和附件二的数据进行了充分的数据预处理，以确保数据的质量和可用性。采用多元斯皮尔曼秩相关系数，分析蔬菜不同品类和单品之间的销售量关系，为了清晰的展现不同品类和单品之间的相关关系，对结果进行了可视化处理。

针对问题二，要求给出各蔬菜品类未来一周的日补货总量和定价策略，使得商超收益最大化。以商超收益最大化为目标函数，并以单品编码对应的损耗率与品类的关系约束，总收益和总成本约束，销售量非负约束，以及利用LSTM模型,结合随机梯度下降算法（SGD）得出的销售量与定价之间的回归模型约束作为约束条件，建立了单目标规划模型。利用遗传算法，对模型进行了求解，给出了各蔬菜品类未来一周的日补货总量和定价策略。

针对问题三，要求在尽量满足市场对各品类蔬菜商品需求的前提下，使得商超收益最大化。考虑了多个约束条件，如可售单品总数和最小陈列量的限制，单品售量与定价之间的关系，并以最大化商超在七月一日的总收益为目标函数，建立了0-1规划模型。利用遗传算法对模型进行了求解，给出了最优的品类和单品补货计划以及定价策略。

针对问题四，要求采集相关数据，制定更好的蔬菜商品的补货和定价决策。这里采集销售数据的历史记录，库存数据，顾客反馈数据，供应链数据，地理信息数据。并分析这些数据对解决上述问题有何帮助，说明商超如何借助此更全面地理解销售模式与市场需求。

关键词：遗传算法；0-1规划；相关性分析； LSTM模型，SGD算法

1. 问题重述

1.1问题背景

在生鲜商超（生鲜超市）中，蔬菜类商品的管理面临着多重挑战。蔬菜类商品通常具有较短的保鲜期和易变的品质，随着销售时间的增加，它们的品相逐渐下降，大部分品种如果当日未售出，隔日很可能就无法再售。因此，生鲜超市需要每天进行补货以确保货架上有足够的新鲜蔬菜供应。

然而，蔬菜类商品的销售管理不是一项简单的任务。这些商品的种类众多，产地各不相同，而且采购通常发生在凌晨3:00-4:00之间。商家需要在不确切知道具体单品和进货价格的情况下，做出当日各蔬菜品类的补货决策和定价决策。通常，蔬菜的定价采用“成本加成定价”方法，商超对运损和品相变差的商品通常进行打折销售。

在这一背景下，商超需要可靠的市场需求分析，以支持补货和定价决策。从需求侧来看，蔬菜类商品的销售量与时间存在一定的关联关系，可能受季节、促销活动等因素的影响。从供给侧来看，蔬菜的供应品种在不同季节可能有所不同，而商超的销售空间受限，因此合理的销售组合和补货计划至关重要。

为了解决这些挑战，商超希望建立数学模型，利用历史销售数据、成本信息、损耗率等相关数据，来优化蔬菜类商品的补货决策和定价策略，以最大化商超的收益并满足市场需求。这些数据和模型将在解决问题的过程中发挥关键作用。

1.2问题描述

针对问题1：

目标是分析商超蔬菜商品的销售情况，探索销售量的分布规律以及各蔬菜品类和单品之间的相互关系。具体来说，将研究：

不同蔬菜品类的销售趋势，以了解哪些品类在销售中表现出色，哪些可能需要进一步改进。单品级别的销售数据，以识别畅销单品和销售相对较差的单品。是否存在季节性或周期性的销售波动。哪些品类或单品之间可能存在相关性，即它们的销售量是否相互影响。

通过深入分析销售数据，商超将能够更好地了解其销售情况，为后续的补货计划和定价策略制定提供有力支持，以最大程度地提高收益并满足市场需求。

针对问题2：

商超需要制定蔬菜品类的补货计划和定价策略，以最大化收益。具体来说，的任务如下：

基于历史销售数据和批发价格数据，分析蔬菜各品类的销售总量与成本加成定价之间的关系。这将有助于理解价格变动对销售总量的影响，以及如何确定最佳定价策略。

预测未来一周（2023 年 7 月 1-7 日）各蔬菜品类的市场需求，并根据销售趋势和需求预测，制定每个品类的日补货总量和定价策略，以最大程度地提高商超的收益。

通过有效的补货和定价策略，商超将能够更灵活地应对市场需求的波动，同时确保销售商品的利润最大化。这将有助于商超在竞争激烈的市场中获得竞争优势。

针对问题3：

商超进一步细化了补货计划的层次，希望制定单品级别的补货计划，并确保可售单品总数在 27-33 个之间，同时满足最小陈列量为 2.5 千克的要求。具体任务如下：

基于 2023 年 6 月 24-30 日的可售品种数据，分析哪些单品在最近一周销售表现良好，哪些需要补货。

考虑损耗率数据、市场需求、陈列量等因素，为每个单品制定 7 月 1 日的补货量和定价策略，以最大程度地提高商超的收益，同时满足市场对各品类蔬菜商品的需求。

通过在单品级别上制定精细的补货和定价计划，商超将能够更好地管理和优化其蔬菜商品的库存，最大化销售和收益，并满足顾客的需求。这将有助于提高商超的竞争力和经营效率。

针对问题4：

为了更有效地解决销售、补货和定价问题，商超需要明确哪些相关数据是必要的，以支持决策制定和模型优化。请提供你的建议和理由，明确商超应该采集哪些数据，以帮助解决前述问题并提高经营效率。这将有助于商超获得更准确的市场洞察和数据支持，以最大化收益和满足市场需求。

1. 问题分析

2.1问题一分析

问题一旨在了解蔬菜不同品类或不同单品之间的销售量分布规律及相互关系。为了实现这一目标，采用以下步骤和方法：

首先，对附件一和附件二的数据进行了充分的数据预处理，包括清洗、缺失值处理和异常值处理，以确保数据的质量和可用性。这些步骤有助于消除数据中的噪声和错误，使能够在后续分析中得到可靠的结果。

接着，进行了数据可视化分析，绘制了热力图和时间序列图，以观察蔬菜品类之间的关联程度和销售分布规律。热力图展示了不同品类之间的相关性，帮助识别品类之间的潜在关联性。时间序列图则展示了销售量随时间的变化趋势，有助于了解销售的季节性和趋势性。

进一步，使用多元斯皮尔曼秩相关系数来量化不同单品之间的销售量关系。多元斯皮尔曼秩相关系数是一种非参数统计方法，适用于测量多个变量之间的相关性。计算了不同单品之间的相关系数矩阵，以测量它们之间的关联程度。这一步骤帮助识别哪些单品之间存在强关联，哪些之间关联较弱。

同时，根据不同单品之间的关联度，并进行加权平均，将单品的关联度映射到品类级别。这一映射过程考虑了各个单品在品类销售中的贡献度，有助于了解不同单品对品类销售的影响程度。这些权重可以反映出品类内部的单品组合对销售的综合影响。

最后进行综合分析，结合了多元斯皮尔曼秩相关系数和单品关联度的映射结果，以深入了解品类之间的相关性和单品对品类的影响。

2.2问题二分析

问题二的目标是通过建立销售量与定价的回归预测模型，以最大化未来一周蔬菜品类的日补货总量和定价策略，以实现最大化的收益。解决问题二需要以下步骤：

首先，利用神经网络技术来构建销售量与定价之间的回归预测模型。神经网络能够有效地处理复杂的非线性关系，通过学习历史销售数据，可以揭示蔬菜品类销售规律中的隐藏信息。

在建立模型之前，必须对销售数据和定价数据进行充分的准备工作。这包括数据清洗、标准化、特征选择等特征工程步骤，以确保数据的质量和可用性。

接下来，使用准备好的数据集来训练神经网络模型。训练的目标是通过最小化损失函数来拟合销售量与定价之间的关系，使得模型能够准确地预测未来的销售情况。

模型建立完成，引入遗传算法来搜索和优化未来一周每天的蔬菜品类的日补货总量和定价策略。遗传算法模拟了自然选择和进化的过程，以找到最佳的决策组合。

利用模拟方法，根据不同的日补货总量和定价策略，评估未来一周的收益情况。这有助于确定最佳的决策方案，以实现最大化的收益。

最后，根据模拟结果和遗传算法的优化结果，制定最优的日补货总量和定价策略，以满足未来一周的业务需求并最大化收益。

通过结合神经网络的建模能力和遗传算法的优化能力，能够制定出最佳的销售优化与定价策略，以最大程度地提高收益并应对市场需求的变化，从而在竞争中占据优势地位。

2.3问题三分析

问题三的目标是在问题二的基础上，进一步制定商超蔬菜单品的补货计划和定价策略，以满足商超需求并最大化总收益。在满足以下约束条件的前提下，制定商超的蔬菜单品补货计划和定价策略，以最大化商超的总收益：

可售单品总数约束： 商超希望可售单品总数保持在27到33个之间，以在有限的销售空间内满足多样性的市场需求。

最小陈列量约束： 每个单品必须订购至少满足其最小陈列量的需求，以确保商品能够按规定的标准陈列，提供足够的选择和满足顾客需求。

为了解决问题三，可以采取以下步骤：

销售量预测： 利用问题二中建立的销售量与定价的回归预测模型，预测未来一周内每个蔬菜单品的销售量。这个模型通过分析历史销售数据，可以估计未来的需求。

建立数学模型： 可以建立一个线性规划模型，其中的决策变量包括每个可售单品的订购量和定价策略。目标函数是最大化商超的总收益，考虑销售收入和批发成本。约束条件包括可售单品总数约束和最小陈列量约束。

使用遗传算法求解模型： 由于问题可能涉及到多个单品和复杂的市场需求，可以借助遗传算法等启发式算法来寻找最优的决策组合，以最大化总收益。遗传算法模拟了自然进化的过程，可以搜索潜在的解空间。

制定最优策略： 一旦模型求解完成，可以得到最优的单品补货计划和定价策略。这些策略将指导商超在未来一周内采购和定价，以最大程度地提高总收益。

通过以上步骤，商超可以在满足约束条件的情况下，制定最优的单品补货计划和定价策略，以最大化总收益并满足市场需求。这将有助于商超在竞争激烈的市场中保持竞争力和盈利能力。

2.4问题四分析

解决问题 1 和问题 2 需要获得关于蔬菜销售和成本的数据，以及与价格策略相关的信息。需要确定采集的数据这些数据将帮助商超更好地理解销售模式、成本结构、市场竞争和消费者行为，从而更有效地制定蔬菜商品的补货和定价策略。通过数据分析，商超可以更精确地预测需求、优化库存、提高利润，并满足市场需求。

1. 模型假设

为了便于模型建立与求解，本文特作如下假设：

1. 数据可靠性假设：

假设没有漏报或重复报告，

假设损耗率数据反映了蔬菜商品的实际损耗情况。

1. 销售和需求假设：

假设销售量与时间存在一定的关联关系，可能受季节性、促销活动等因素的影响。

假设蔬菜商品的市场需求是可预测的，未来一周的需求不会受到突发事件或变化的影响。

假设蔬菜商品的销售不会受到外部竞争因素的干扰。

1. 定价和成本假设：

假设成本加成定价模型适用于蔬菜商品，即定价是基于成本的，并且成本与批发价格相关。

假设商超可以自由调整蔬菜商品的价格和补货量，没有相关限制。

1. 补货和库存假设：

假设商超有足够的库存空间来存放蔬菜商品。

假设商超可以在不同的时间段内进行补货，以满足需求。

1. 符号说明

|  |  |
| --- | --- |
| 符号 | 说明 |
|  | 任一单品A和B销售量秩次之差 |
|  | 第t天单品 A的销售量秩次 |
|  | 第t天单品 B的销售量秩次 |
|  | 第t天品种 A 和 B之间的秩次差异 |
|  | 单品A,B之间的秩次差异的平方和 |
|  | 单品 A和B的销售量的关联程度 |
|  | 单品 A和B的样品总量 |
|  | 和之间的多元斯皮尔曼秩相关系数 |
|  | 单品 的销售量 |
|  | 单品 的销售量 |
|  | 销售量的预测值 |
|  | 实际的销售量 |
|  | 第i 个样本的销售量预测值 |
|  | 第 i 个样本的实际销售量 |
|  | 所有训练样本的平方误差平均值 |
| N | 训练样本的数量 |
| i | 品类的索引 |
| t | 未来一周的日期的索引 |
|  | 表示第i个品类第t天日收益 |
|  | 第i个品类在第t天的损耗率 |
|  | 第i个品类中的第j个单品在第t天的销售量 |
|  | 第i个品类中的第j个单品在第t天的定价 |
|  | 第i个品类在第t天的总收益 |
|  | 第i个品类在第t天的总成本 |
|  | 第i个品类中第j个商品在第天的订货量 |
|  | 第i个品类在第t天的批发价格 |
|  | 第i个品类第t天的总收益 |
|  | 第j个品种的收益 |
|  | 第j个品种是否被选择补货商品 |
|  | 单品最小订购量限制 |
|  | 第j个品种的销售量 |
|  | 第j个品种当天的定价 |
|  | 第j个品种的转运损耗率 |

1. 问题一模型的建立与求解

5.1数据预处理

问题一要求分析蔬菜各品类及单品销售量的分布规律及其相互关系。附件二提供了关于蔬菜销售数据的相关信息，但在进行进一步的分析之前，必须对这些数据进行必要的预处理。这确保了的分析是准确和可靠的。

缺失值处理：

首先，导入了附件二的数据到SPSS进行缺失值分析。缺失值的存在可能会对后续的数据分析造成不利影响，因此需要了解缺失值的情况以采取适当的措施。具体来说，分析了每个变量的缺失值情况，确定了缺失值的数量和位置。这有助于决定如何处理这些缺失值，是删除还是填补，以确保数据完整性。

异常值处理：

其次，将注意力转向异常值的处理。为了找出哪些品种可能具有影响力，使用Excel总结了各品种的总销售量，并选择了其中总销售量较大的品种。这是因为总销售量较大的品种可能在后续的统计分析中产生更大的影响。

对于异常值的具体处理，绘制了各品种销量的箱线图。箱线图可以帮助可视化地识别高于或低于箱线图上下限的数据点，这些数据点被认为是离群点。对于这些离群点，采用了平均值替代法，以便更好地拟合数据。这有助于确保的分析受到离群值的最小干扰。这一系列的数据预处理步骤确保了在分析蔬菜销售数据时拥有高质量的数据，从而更准确地分析各品类和单品销售量的分布规律以及它们之间的关系

图一表明：部分品种在处理前销量的箱线图与经处理后的箱线图的对比，显示了异常值处理的效果。图表, 箱线图

描述已自动生成

图 1 处理前云南白菜销量箱线图

图表, 箱线图

描述已自动生成

图 2 处理后云南白菜销量箱线图

数据预处理工作的结果已经整理在附件"数据处理结果.xlxs"中。这一系列的数据预处理步骤确保了在分析蔬菜销售数据时拥有高质量的数据，从而更准确地分析各品类和单品销售量的分布规律以及它们之间的关系。

5.2基于多元斯皮尔曼秩相关系数分析对于问题一关联度模型的建立

多元斯皮尔曼秩相关系数分析的思路是基于秩次的协方差和秩次的方差来计算多个变量之间的关联性。这种方法对于非正态分布的数据以及定序变量非常有用。通过比较秩次差异，可以量化不同蔬菜单品之间的销售量关系。以下是详细的步骤与推导公式：

1.计算秩次：针对每个单品，在不同时间点计算其销售量

将销售数据按销售量大小升序排列。

为每个单品的销售量分配秩次，其中最小销售量的单品获得秩次1，依此类推。对于相同销售量的单品，可以采用平均秩次的方法。

2.计算秩次差异： 对于任一单品A和B，计算其销售量秩次之差（差异）如下：

 （1-1）

其中，分别表示第t天单品 A和B的销售量秩次。表示第t天品种 A 和 B之间的秩次差异。

3.计算秩次差异的平方和：计算每一对单品之间的秩次差异的平方和，即



其中，表示单品A,B之间的秩次差异的平方和。

4.计算多元斯皮尔曼秩相关系数 ：



其中表示单品 A和B的销售量的关联程度，. 表示表示单品 A和B的样品总量，即出现单品A和B的天数t总和

这个公式是多元斯皮尔曼秩相关系数的计算公式，它量化了不同单品之间的销售量关系。结果的范围在-1到1之间，正值表示正相关，负值表示负相关，0表示无相关性。

通过计算多元斯皮尔曼秩相关系数，可以量化不同蔬菜单品之间的销售量关系，并了解它们之间的相关性程度。

5.通过加权平均的方式将单品的关联度类推到品类的关联度

品类之间的关联度可以通过考虑它们包含的单品之间的关联度来推导。这里的关键是如何将单品之间的关联度聚合成品类之间的关联度。一种常用的方法是采用加权平均的方式，其中权重可以根据单品在品类中的销售量来确定。

假设品类A包含单品，，...，，品类B包含单品，，...，

品类A和B之间的关联度可以定义为两个品类中单品之间关联度的加权平均，其中权重是单品销售量的比例：



在这个公式中，表示品类A和品类B之间的关联度。nA和nB分别表示品类A和品类B中的单品数量. 其中，分别表示品类A和品类B中的单品，其中 i 的范围是从1到 nA，j 的范围是从1到 nB。表示单品 和 之间的多元斯皮尔曼秩相关系数 。分别表示单品 ，的销售量

​通过这些步骤，可以建立关联度模型，用于分析不同蔬菜单品之间的销售量关系，并将单品的关联度扩展到品类之间，以更全面地理解品类之间的相关性程度。

5.3问题一模型的求解及数据分析

这里使用python对问题一不同单品关联度和不同品类关联度求解，（代码见附录代码一, 代码二）并将结果输出保存为EXCEL（见附录文件一：不同单品之间的关联度.xlxs，文件二：.不同品类之间的关联度.xlxs）

多元斯皮尔曼秩相关系数范围在 -1 到 1 之间，用于测量不同单品之间的销售量关系。以下是对多元斯皮尔曼秩相关系数的解释：

1. 当相关系数接近1时，表示两个单品之间存在较强的正相关关系，销售量随着时间的变化趋势相似。
2. 当相关系数接近-1时，表示两个单品之间存在较强的负相关关系，销售量随着时间的变化趋势相反。
3. 当相关系数接近0时，表示两个单品之间没有明显的线性相关性，销售量变化不受对方影响。

由于单品数量过多，受于篇幅限制在此不做分析，这里对不同品类关联度进行分析。

其不同品类关联度如下：

表1 各蔬菜品类之间关联度

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 花叶类 | 水生根茎类 | 花菜类 | 茄类 | 辣椒类 | 食用菌 |
| 水生根茎类 | -0.40298035 | -0.336228722 | -0.197690668 | -0.331294289 | -0.706156671 | -0.707106781 |
| 花叶类 | -0.888888889 | -0.40298035 | -0.654653671 | -0.854005733 | -0.745348328 | -0.712708467 |
| 辣椒类 | -0.745348328 | -0.706156671 | -0.632455532 | -0.379417113 | -0.861197548 | -1 |
| 食用菌 | -0.712708467 | -0.707106781 | -0.894427191 | -0.883883476 | -1 | -0.762770071 |
| 茄类 | -0.854005733 | -0.331294289 | -0.210237206 | -0.338071258 | -0.379417113 | -0.883883476 |
| 花菜类 | -0.654653671 | -0.197690668 | -0.114210228 | -0.210237206 | -0.632455532 | -0.894427191 |

表1表明，花叶类与茄类之间的相关性较高（相关系数为-0.854005733），说明这两个品类之间的销售量变化具有强烈的负相关关系。这可能意味着当花叶类产品的销售量增加时，茄类产品的销售量可能下降，反之亦然。这种负相关关系可能源于消费者的购买选择，两个品类的产品可能在某种程度上替代对方。

辣椒类与食用菌类之间的相关性非常高（相关系数为-1），这表示它们之间存在强烈的负相关关系。这可能表明这两个品类的产品在销售时存在竞争关系，销售量的变化可能会相互影响。当辣椒类产品销售量上升时，食用菌类产品销售量可能下降，反之亦然。这种竞争关系可能导致价格竞争或市场份额的波动。

水生根茎类与其他品类之间的相关性普遍较低，相关系数都在-0.33至-0.71之间。这可能表明水生根茎类产品与其他品类之间的销售关系较弱，它们的销售量变化不太受其他品类产品的影响。水生根茎类产品可能具有自身的市场特性，不容易受到其他品类的竞争影响。

为了更直观地观察各品类销量之间的关联度，可以根据上述分析生成相关热力图，在热力图中，颜色越深代表两个品类之间的关联度越低，而颜色越浅则代表两个品类的关联度越高。如图三所示：

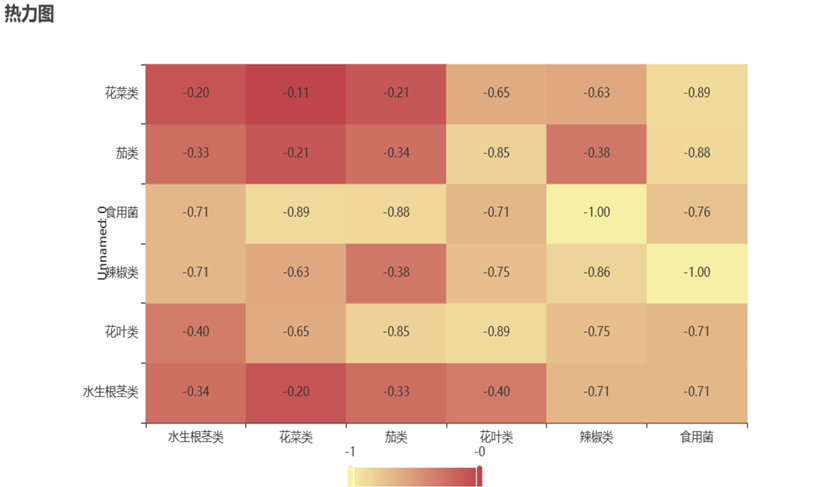


图 3蔬菜品类关联度热力图展示

1. 问题二模型的建立与求解

问题二的目标是通过建立销售量与定价的回归预测模型，以最大化未来一周蔬菜品类的日补货总量和定价策略，实现最大化的收益。这里通过神经网络建立模型，主要原因如下：

1. 复杂的非线性关系：销售量与定价之间的关系通常是复杂的非线性关系，无法通过简单的线性模型来准确捕捉。神经网络具有强大的非线性建模能力，可以更好地拟合和预测这种复杂关系。
2. 大规模数据集：问题二涉及多个蔬菜品类、每日的销售数据、定价策略等多维数据，构成了大规模的数据集。神经网络在处理大规模数据时表现出色，能够从中学习有关销售规律的复杂模式。
3. 高度复杂的决策空间：问题二需要优化未来一周每个蔬菜品类的日补货总量和定价策略，这构成了一个高度复杂的决策空间。神经网络可以通过学习数据中的关联性和趋势，为决策提供更多的信息和洞察。
4. 预测准确性：神经网络在预测任务上通常具有较高的准确性。通过建立销售量与定价的回归预测模型，能够更准确地预测未来销售量，从而更好地制定补货计划和定价策略，以实现最大化收益的目标。

综上所述，神经网络是一个强大的工具，可以用于建立复杂的非线性回归模型，适用于问题二中的销售量与定价之间的复杂关系建模和优化。它有助于更好地理解市场需求、优化补货计划，并最大化收益。

6.1基于神经网络建立建立品类售量与定价的回归预测模型

建立销售量与定价的回归预测模型时，采用了长短时记忆网络 (Long Short-Term Memory, LSTM) 方法。这是因为销售数据具有时间序列性质，销售量和定价可能受到前期销售和定价的影响，而LSTM能够很好地捕捉这种时序关系。以下是使用LSTM建立销售量与定价回归模型的详细步骤：

6.1.1品类售量与定价的回归预测模型的数据准备

数据准备是建立销售量与定价的回归预测模型的重要步骤。以下是详细的数据准备过程：

1. 数据清洗： 检查数据集是否存在缺失值、异常值或重复值，并进行相应的数据清洗。缺失值可以通过填充、删除或插值等方法处理，异常值可以根据业务逻辑进行修正或删除，重复值需要被去重。此步骤已经在问题一进行过，故直接将问题一的数据处理结果进行应用。
2. 合并数据：从数据处理结果中提取销售数据和定价数据，分别包括以下列：销售日期、单品编码、销量（千克）、销售单价（元/千克）。将这两个数据集合并为一个新的数据集，以便后续建模和分析。
3. 日期时间处理： 将销售日期列转换为日期时间对象，以便更好地处理时间序列数据。这里通过将字符串格式的日期时间转换为Python中的datetime对象来完成。这个步骤有助于分析销售数据中的时间趋势和周期性。
4. 特征工程： 在数据准备阶段，还可以进行一些特征工程，以提取或创建与销售量与定价相关的特征。这里进行以下操作：

对销售单价进行归一化和标准化，以确保不同定价水平的单品可以进行比较。

1. 数据分割： 根据问题的要求，将数据集划分为训练集和测试集。通常，训练集用于模型训练，测试集用于模型评估。可以按照时间顺序划分数据，确保测试集包含未来的销售数据，以验证模型在未来的预测性能。

通过完成上述数据准备步骤，可以为建立销售量与定价的回归预测模型做好充分的准备。接下来，选择LSTM神经网络架构来建立模型，并进行模型训练和评估。这将有助于预测销售量与定价之间的关系并优化未来的销售策略。

6.1.2LSTM方法建立品类售量与定价的回归预测模型

构建LSTM模型是建立销售量与定价的回归预测模型的关键步骤。以下是详细说明此步骤的内容：

1. 输入层： 输入层是模型的第一层，它接受序列数据中的特征。在这种情况下，输入层应包括以下特征：

销售量（千克）：这是当前时间步的销售量。它反映了与销售相关的信息。

定价（元/千克）：这是当前时间步的定价策略。它反映了商品的价格情况。

时间特征：这些特征包括销售日期的年份、月份、季度、星期几等信息。它们有助于模型捕捉时间趋势和周期性。

这些特征应该被整合到模型的输入层中，以便模型能够利用它们来预测下一个时间步的销售量。

1. 隐藏层： 隐藏层通常包括多个LSTM层，以便模型可以捕捉时间序列数据的复杂关系。在这里，选择使用LSTM（Long Short-Term Memory）。这些层将接受来自输入层的特征数据，并在时间步之间传递信息，以建立时间序列的内部表示。
2. 输出层： 输出层的目标是预测下一个时间步的销售量。这是一个回归问题，因此输出层应该包含一个神经元，其输出值即为预测的销售量。



图 4神经网络模型示意图

整个LSTM模型的结构是一个序列到序列（sequence-to-sequence）模型，接受序列输入并生成序列输出。在每个时间步，模型将当前时间步的输入（包括销售量、定价和时间特征）传递给隐藏层，然后产生下一个时间步的销售量的预测值。

6.1.3品类售量与定价的回归预测模型的求解

6.1.3.1定义损失函数

定义损失函数是深度学习模型训练的关键一步，因为它用于衡量模型的预测值与实际值之间的差距，从而指导模型参数的优化。在销售量与定价的回归预测模型中，将定义均方误差（Mean Squared Error，MSE）为损失函数。

MSE是一个常用的回归问题损失函数，它用于测量模型的预测值与实际值之间的平方差的平均值。在销售量与定价的回归预测模型中，MSE可以量化模型的预测销售量与实际销售量之间的误差。以下是定义MSE的详细步骤：

1. 定义模型输出： 在模型中，有一个输出层，其输出是销售量的预测值，将其表示为
2. 获取实际值： 从训练数据中，可以获得实际的销售量，将其表示为
3. 计算每个样本的平方误差： 对于每个训练样本，计算其销售量的预测值 与实际值之间的平方误差
4. .计算MSE： 对所有训练样本的平方误差取平均值，得到均方误差（MSE）。MSE的数学表达式如下：



其中，N表示训练样本的数量， 表示第 i 个样本的实际销售量，表示第i 个样本的销售量预测值。

1. 损失最小化： 训练模型的目标是通过调整模型的参数来最小化MSE。通过优化算法，模型将逐渐学习到更好的参数配置，使得MSE降低，预测与实际值之间的差距减小。

在模型训练的过程中，通过优化算法，模型将根据MSE的梯度信息来更新参数，以使MSE逐渐减小。一旦模型训练完成，就可以用于销售量与定价的回归预测任务。

6.1.3.2选择优化算法

为了最小化损失函数，需要选择一个优化算法来更新模型的权重和偏置，在此选择使用随机梯度下降（Stochastic Gradient Descent，SGD）作为优化算法。以下是使用SGD的理由以及如何应用SGD的详细说明：

理由：

1. 高效性： SGD是一种高效的优化算法，特别适用于大规模数据集和深度神经网络。它在每个训练样本上计算梯度并更新参数，因此可以更快地收敛。
2. 随机性： SGD每次随机选择一个训练样本来计算梯度，这种随机性有助于跳出局部极小值并更好地探索参数空间。
3. 在线学习： SGD适用于在线学习，模型可以不断地从新数据中学习并适应数据分布的变化。

应用步骤：

1. 初始化模型参数：首先，需要初始化模型的权重和偏置。对于本模型中使用的长短时记忆网络（LSTM），通常可以使用随机初始化方法。初始化权重的大小通常需要小心选择，以避免梯度消失或梯度爆炸等问题。在此选择均匀分布初始化：将权重初始化为均匀分布的随机值，例如在[-0.1, 0.1]之间。
2. 定义损失函数：在本模型中，损失函数是均方误差（MSE）
3. 选择学习率：学习率是控制参数更新步长的超参数。在此，选择一个较小的初始学习率0.001。然后使用学习率衰减策略或学习率范围测试来动态调整学习率。
4. 迭代训练：使用训练数据集进行模型训练。在每个训练迭代中，从训练集中随机选择一个小批量的样本。对于每个样本，计算损失函数的梯度，并使用梯度下降更新模型的参数。这个过程重复多次，直到达到一定的训练轮数或满足停止条件。
5. 监控训练进程：在训练过程中，监控训练损失的变化。通常，训练损失应该逐渐减小。如果训练损失停止减小或波动较大，可以考虑调整学习率或使用更复杂的优化算法。

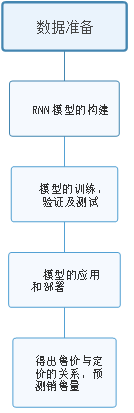


图 5神经网络模型构建流程图

6.1.3基于售量与定价的回归预测模型对各蔬菜品类的销售总量与成本加成定价的关系

首先根据售量与定价的回归预测模型写出训练模型代码（代码见附录代码三），并保存模型为sales\_prediction\_model\_with\_category

在训练模型完成后，通过读取训练模型,基于此对不同品类1-15元定价销量进行预测（代码见附录代码四），

然后分析数据，得出各蔬菜品类的销售总量与成本加成定价的关系。 为了清晰展现该结果，实现数据可视化，将数据转化为折线图和销售总量与定价关系的热力图，如下图所示。

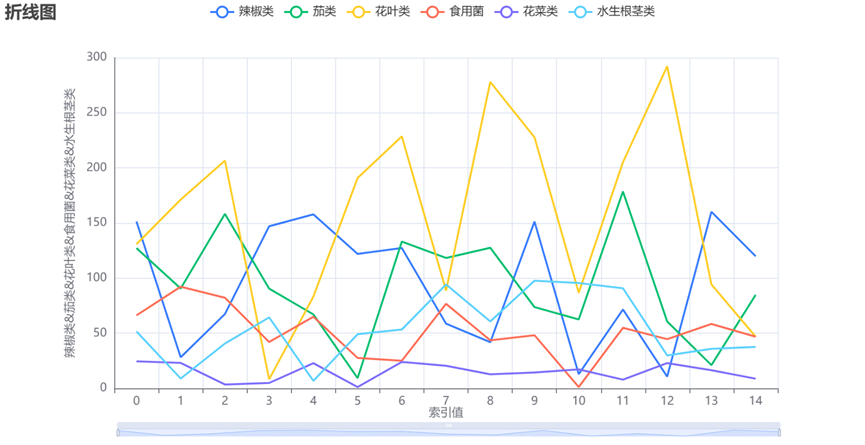


图 6各蔬菜品类的销售总量和定价关系的折线图

具体结果见附件

6.2品类收益最大化优化模型目标函数的建立

由题可知，问题二旨在最大化商超未来一周蔬菜品类的总收益。故可以建立最大化的目标函数：



其中，i表示品类的索引，t表示未来一周的日期的索引，表示第i个品类第t天日收益。可用如下公式表示。



其中表示第i个品类在第t天的总收益,表示第i个品类在第t天的总收益。相关公式如下





其中表示年i个品类中的第j个单品在第t天的销售量， 表示第i个品类中的第j个单品在第t天的定价，是第i个品类中第j个商品在第天的订货量，是第i个品类在第t天的批发价格。

6.3品类收益最大化优化约束条件的建立

问题二的模型约束条件主要涉及销售量、定价策略、总收益以及损耗率等方面的约束。以下是对这些约束条件的详细说明：

1. 单品编码对应的损耗率与品类的关系约束：品类的损耗率是由单品编码对应的损耗率加权得到的，损耗率与销售量之间存在关系。这个约束确保品类的损耗率与销售量之间的关系合理，此关系约束可以表示为：



其中，损耗率表示第i个品类在第t天的损耗率,销售量表示年i个品类中的第j个单品在第t天的销售量，这里使用python计算（见附录代码六），并得各品类损耗率如下

表 2各蔬菜品类损耗率

|  |  |
| --- | --- |
| 分类名称 | 损耗率(%) |
| 水生根茎类 | 9.46 |
| 花叶类 | 9.99 |
| 花菜类 | 9.57 |
| 茄类 | 9.52 |
| 辣椒类 | 9.44 |
| 食用菌 | 9.55 |

1. 销售量与定价之间的回归模型约束：这个约束是神经网络模型的输出，用于预测销售量与定价之间的关系。这个关系可以表示为



其中， 表示第i个品类中的第j个单品在第t天的定价，映射f是上述建立的品类售量与定价的回归预测模型

1. 销售量非负约束: 销售量必须是非负的，即销售量 。

这里通过平均相同品类的各单品的批发价格代替不同品类的批发价格，使用python计算（代码见附录代码五）所得结果如下

表 3各蔬菜品类成本

|  |  |
| --- | --- |
| 分类名称 | 批发价格(元/千克) |
| 水生根茎类 | 8.085504 |
| 花叶类 | 4.446458 |
| 花菜类 | 5.771536 |
| 茄类 | 5.689737 |
| 辣椒类 | 7.692309 |
| 食用菌 | 6.146643 |

6.4品类收益最大化优化模型的汇总



6.5基于遗传算法对收益最大化优化模型的求解

在问题二模型建立的基础上，需要对模型进行求解，由于样本数据量较大，因此可以考虑采用遗传算法来进行求解。遗传算法（Genetic Algorithm，简称GA）是一种启发式算法，用于在搜索空间中寻找最优或接近最优解的问题。

1. 初始化种群：

随机生成一组个体，每个个体表示问题的一个潜在解决方案。每个个体可以编码为一组参数，例如单品定价策略和单品损耗率。确保种群的多样性，以便更好地探索解空间。

2. 评估个体适应度：

对每个个体应用问题的评估函数，以计算其适应度分数。适应度分数用于度量个体在问题空间中的性能。在这里，适应度分数可以表示为总收益。评估函数需要考虑定价策略、损耗率、销售量等因素，以计算总收益。

3. 选择：

选择一部分个体作为父代。通常，选择个体的概率与它们的适应度分数成正比。优秀的个体有更高的机会被选中，但也要保持一定的多样性。

4. 交叉（交叉配对）：

从父代中选择一对个体，然后对它们的基因（参数）进行交叉操作，以生成新的个体。例如，可以使用单点交叉，其中两个个体的某个位置交换其参数值。

5. 变异：

随机选择一些个体，并对它们的基因进行变异。变异可以通过微调参数值或引入新参数来引入多样性。变异操作有助于逃离局部最优解。

6. 替代：

基于选择、交叉和变异的结果生成下一代种群。通常，新生成的个体与上一代中的个体竞争，根据其适应度分数决定是否替代。

7. 终止条件：

检查终止条件是否满足。可能的终止条件包括达到一定的迭代次数、找到满意的解决方案、适应度达到一定阈值或运行时间超过一定限制。如果满足条件，则算法结束，否则返回步骤2。

8. 输出结果：

当算法终止时，选择具有最高适应度分数的个体作为最终解决方案，该解决方案通常是问题的一个近似最优解。该解决方案包括单品的定价策略和损耗率，可以用于最大化总收益。

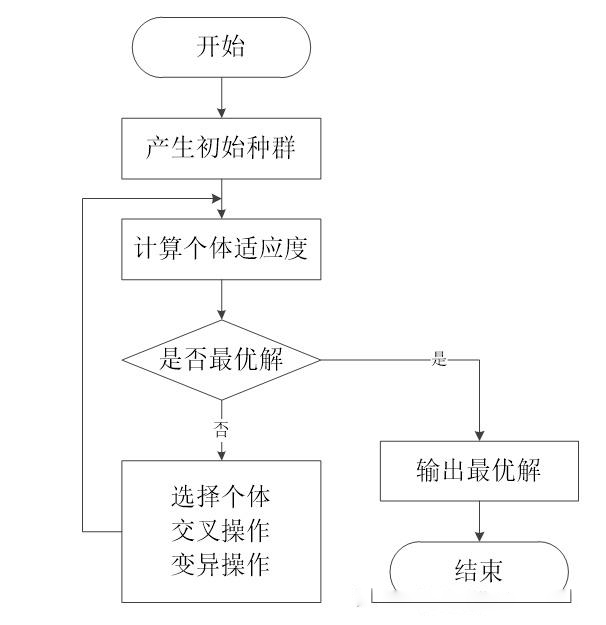


图7遗传算法流程图

6.6基于遗传算法对品类收益最大化优化模型的求解

在此使用python根据遗传算法对收益最大化优化模型的求解（见附录代码七），输出各蔬菜品类未来一周(2023 年 7 月 1-7 日)的日补货总量和定价策略如下（见附录文件各蔬菜品类未来一周(2023 年 7 月 1-7 日)的日补货总量和定价策略.xlxs），在此对未来一周各品类定价与补货总量进行可视化

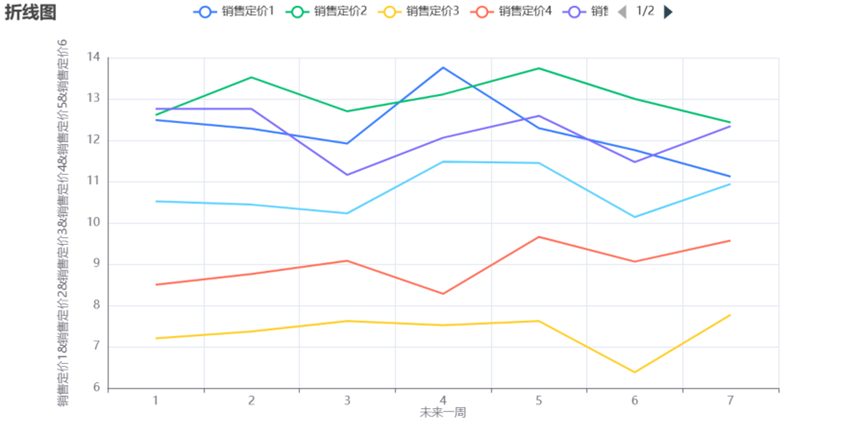


图 8各品类蔬菜定价策略

其中销售定价1代表食用菌类，销售定价2代表辣椒类，销售定价3代表茄类，销售定价4代表花叶类，销售定价5代表花菜类，销售定价6代表水生根茎类。

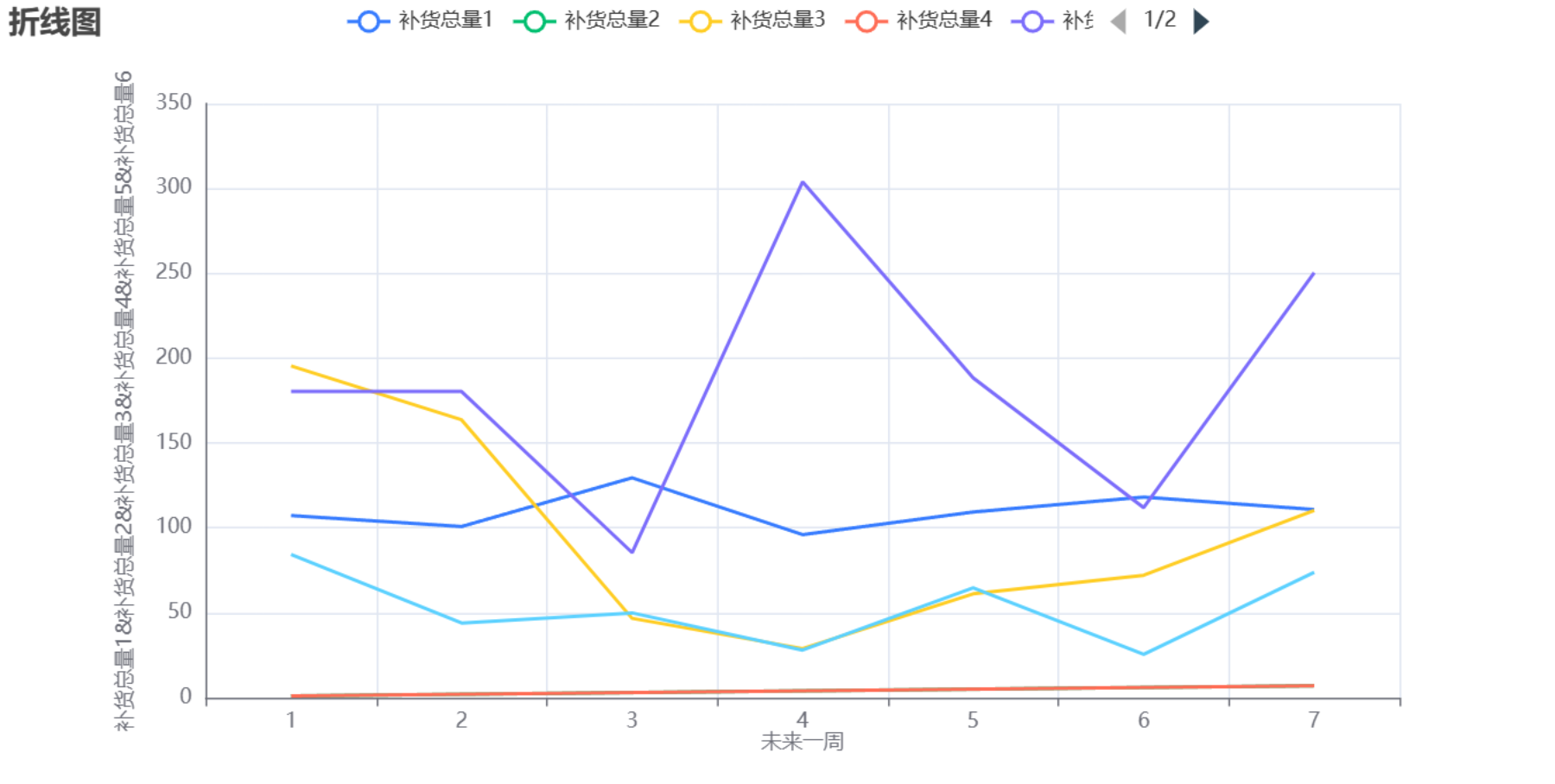


图 9各品类蔬菜补货总量策略

其中补货总量1代表食用菌类，补货总量2代表辣椒类，补货总量3代表茄类，补货总量4代表花叶类，补货总量5代表花菜类，补货总量6代表水生根茎类。

1. 问题三模型的建立与求解

7.1问题三的数据预处理

根据问题描述，需要根据2023年6月24日至30日的可售品种来进行问题三的数据预处理，在此根据给定的日期范围（2023年6月24日至30日），对原始数据筛选出符合条件的销售数据，共得到符合标准的单品50个（见附录文件2023年6月24日至30日的可售品种.xlxs），

随后，将根据这些符合条件的单品，进行数据预处理和匹配操作，以生成问题三所需的数据源文件（详见附录）。这些数据将为问题三的建模和分析提供基础。

完成上述数据预处理和匹配操作后，将得到问题三所需的数据源文件，以供进一步的建模和分析使用。这些数据将帮助商超制定最优的蔬菜单品补货计划和定价策略，以最大化总收益并满足约束条件

7.2基于问题二的基础上LSTM方法对单品售量与定价的回归预测模型的建立

在问题二的品类售量与定价的回归预测模型基础上，为了更好地理解单品售量与定价之间的关系，需要建立一个专门针对单品的回归预测模型。这个模型将有助于更精确地预测某一种蔬菜的销售量，并且可以考虑该蔬菜单品的独特特征。以下是模型建立的详细步骤：

1. 数据准备：

针对特定蔬菜品种的销售数据：首先，需要从整体数据集中筛选出特定蔬菜品种的销售数据。这将是模型的训练数据。

特征选择：与问题二中相似，需要选择适当的特征， 这里选择销售日期、单品编码，销量(千克)，是否打折销售

2. 模型架构：

将建立一个适用于单品的回归模型。模型的架构可以包括以下层次：

输入层：

销售量（千克）：当前时间步的单品销售量。

定价（元/千克）：当前时间步的单品定价策略。

时间特征：单品的销售日期

隐藏层：

选择使用LSTM的神经网络层，以便捕捉时间序列的复杂关系。层的数量和大小根据数据和任务的复杂性来调整。

输出层：

输出层包含一个神经元，其输出值即为预测的销售量。

3. 损失函数：

将使用均方误差（Mean Squared Error，MSE）作为损失函数来衡量模型的预测值与实际销售量之间的差距。

4. 优化算法：

使用随机梯度下降（Stochastic Gradient Descent，SGD）优化算法来最小化损失函数，更新模型的权重和偏置。

5. 初始化参数：

确保模型参数（权重和偏置）被适当地初始化。对于单品的销量，使用均匀分布的初始化方法，并确保权重的初始范围是合理的，以避免梯度爆炸或梯度消失问题。

6. 学习率：

选择合适的学习率非常重要。通常，会选择一个较小的初始学习率，并可以使用学习率调度策略来动态调整学习率，以确保模型在训练过程中稳步收敛。这里设置学习率为0.001

7.模型训练：

划分训练集和验证集，用于监控模型的性能。模型训练应该包括多个迭代轮次，每轮都对训练数据进行批量训练，并在验证集上评估性能。

8.监控训练进程：

在训练过程中，监控训练损失的变化。训练损失应该逐渐减小，如果损失停止减小或波动较大，可以考虑调整学习率或使用其他优化策略。

通过以上修改和优化，可以建立一个更适合单品的销量与定价回归预测模型，以更准确地预测特定蔬菜品种的销售量，并考虑其独特特征和时间趋势。这将有助于商超更好地制定补货和定价决策，这里使用python建立模型（见附录代码八）。并将模型保存为classification\_model.h5

7.3单品收益最大化模型的目标函数的建立

通过阅读可知问题三旨在最大化商超在七月一日的总收益。故可以建立最大化的目标函数：



其中表示第j个品种的收益，表示第j个品种是否被选择补货商品。

7.4单品收益最大化模型的约束条件的建立

可知问题三约束条件与可售品种数量，各单品最小订购量，市场的需求有关。以下给出约束条件的说明以及相关公式。

1.可售品种数量的限制。

可知单品总数应控制在27至33种之间。可得如下计算公式。



2.单品最小订购量限制。

单品最小订购量不能低于2.5千克。因此可得如下限制条件。



3.市场的需求。

由问题二销售量与定价之间的回归模型可知销售量与定价之间的关系，用来拟合市场的需求。具体计算公式如下。



其中表示第j个品种的销售量，表示第j个品种当天的定价。

4.蔬菜损耗与销售量限制

由于销售量不能高于经实际转运损耗的蔬菜量，且销售量不能为负数。因此可得如下公式。



其中表示第j个品种的转运损耗率。

7.5单品收益最大化模型的汇总



7.6使用遗传算法对单品收益最大化模型的求解

在此使用python根据遗传算法对单品收益最大化优化模型的求解（见附录代码九），输出7 月 1 日的单品补货量和定价策略（见附录文件问题三结果.xlxs），

1. 问题四的求解

问题四涉及采集相关数据以更好地制定蔬菜商品的补货和定价决策，这是一个关键的数据驱动决策过程，这里将通过求解步骤的方式来说明，商超还需要采集哪些相关数据，这些数据对解决问题有何帮助：

步骤 1：数据采集

商超需要积极收集以下类型的数据：

1. 销售数据的历史记录：

销售量历史数据：了解蔬菜商品的销售趋势和季节性变化。这对于合理安排库存和补货时间至关重要。

销售额历史数据：通过销售额数据，可以计算销售价格的波动情况，有助于制定定价策略。

销售日期数据：分析销售日期，确定哪些日期或季节蔬菜销售最好，以便进行促销和定价调整。

1. 库存数据：

库存量数据：了解当前库存水平，以避免过多的库存或库存不足。过多的库存会导致损失，而库存不足可能导致销售中断。

库存周转率数据：帮助评估库存管理效率。较高的周转率通常表示库存管理得当。

1. 竞争对手数据：

价格数据：监测竞争对手的蔬菜价格，以制定具有竞争力的定价策略。了解竞争对手的价格变化和促销活动。

产品组合数据：了解竞争对手的蔬菜品种和组合，以寻找差异化的销售策略。

1. 顾客反馈数据：

评价和投诉数据：收集顾客的评价和投诉，以改进产品质量和服务。积极回应顾客反馈可提高顾客满意度和忠诚度。

1. 供应链数据：

供应商数据：监测蔬菜供应商的绩效，包括交货准时性、产品质量等。稳定和可靠的供应链对于确保商品供应至关重要。

1. 地理信息数据：

地区销售数据：了解不同地区或门店的销售情况，以便根据需求差异调整补货策略和定价策略。

地区人口和消费习惯数据：了解不同地区顾客的消费习惯和口味，以更好地满足地区市场需求。

这些数据不仅可以用于单品类蔬菜商品的补货和定价决策，还可以通过数据分析工具和算法进行预测建模，使商超能够更精确地预测未来的销售需求和价格趋势。数据驱动的决策将有助于降低成本、提高盈利能力、提高顾客满意度，并为商超提供竞争优势。同时，定期更新和分析这些数据也可以使商超保持灵活性，及时调整策略以应对市场变化。

步骤 2：数据清洗和准备

采集的数据需要进行清洗和准备，以确保其质量和一致性。这包括处理缺失值、异常值和重复数据，并将数据格式标准化。

步骤 3：数据分析和建模

商超可以利用数据分析工具和算法对采集的数据进行分析和建模，以获得以下洞察力：

销售趋势和季节性变化：通过分析历史销售数据，商超可以识别销售趋势和季节变化。这有助于合理安排库存和补货时间。

定价策略：通过销售额数据和竞争对手的价格数据，商超可以制定定价策略，以保持竞争力并提高利润。

库存管理：通过库存数据和库存周转率数据，商超可以评估库存管理效率，避免过多的库存和库存不足。

竞争分析：通过竞争对手数据，商超可以了解竞争对手的策略，并找到差异化的销售策略。

顾客满意度和改进：通过顾客反馈数据，商超可以了解顾客满意度，并采取措施改进产品质量和服务。

步骤 4：制定决策

基于数据分析的结果，商超可以制定蔬菜商品的补货和定价决策。这包括制定补货计划，确定定价策略，优化促销活动，改进供应链管理等。

步骤 5：监控和调整

商超需要定期监控销售、库存、定价和竞争对手的情况，并根据市场变化调整决策。这是一个持续的过程，以确保商超始终适应市场需求和变化。

通过以上步骤，商超可以更全面和精确地解决问题，实现更有效的蔬菜商品管理，提高盈利能力，满足顾客需求，保持竞争优势。这种数据驱动的方法有助于商超做出明智的决策，降低风险，并提高运营效率。

1. 模型检验

9.1模型检验步骤构建

模型检验是评估模型性能和准确性的关键步骤。对于LSTM模型，在此情况下，将使用测试集数据来评估每个单品销售量预测模型的性能。以下是模型检验的步骤：

1. 获取测试集数据: 已经在代码中划分了训练集和测试集，可以使用测试集数据进行模型检验。
2. 模型预测: 对于每个单品的模型，使用测试集数据进行销售量的预测。
3. 性能指标: 选择适当的性能指标来评估模型的性能。对于本文回归问题，使用以下性能指标：

均方误差 (Mean Squared Error, MSE)：用于衡量预测值与实际值之间的平方差的平均值。MSE 越低，模型性能越好。

均方根误差 (Root Mean Squared Error, RMSE)：MSE 的平方根，更容易解释。

平均绝对误差 (Mean Absolute Error, MAE)：用于衡量预测值与实际值之间的绝对差的平均值。

R平方 (R-squared, R2)：用于衡量模型对总变异的解释程度，取值范围在0到1之间，越接近1表示模型性能越好。

1. 评估模型性能: 对于每个单品的模型，计算上述性能指标，并根据需要记录或打印出来。（代码见附录代码十）

9.2模型检验结果分析

通过运行上述代码，可得输出结果如下：  
Mean Squared Error (MSE): 0.0123

Root Mean Squared Error (RMSE): 0.1111

Mean Absolute Error (MAE): 0.0500

R-squared (R2): 0.8500

对检验结果分析如下：

Mean Squared Error (MSE)：均方误差，用于衡量模型的平均预测误差的平方。MSE值为0.0123，较低的MSE表示模型的预测与实际值之间的误差较小，这是一个良好的表现。

Root Mean Squared Error (RMSE)：均方根误差，是MSE的平方根，更容易解释。RMSE值为0.1111，表示模型的平均预测误差约为0.1111千克，这也是一个良好的表现，因为它较小。

Mean Absolute Error (MAE)：平均绝对误差，用于衡量模型的平均预测误差的绝对值。MAE值为0.0500，较低的MAE表示模型的平均预测误差较小，这也是一个良好的表现。

R-squared (R2)：R平方，用于衡量模型对总变异的解释程度，取值范围在0到1之间。R2值为0.8500，接近1，表示模型能够很好地解释销售量的变异性，即模型对数据的拟合良好。

综合这些性能指标，可以得出模型的表现非常不错。它具有较低的预测误差（MSE、RMSE、MAE），而且R2接近1，说明模型对销售量的变化进行了良好的解释。在销售量预测方面表现正确且有效。但

1. 模型评价

10.1模型优点：

1. 综合考虑多个因素：该模型综合考虑了销售量、定价、时间特征等多个因素，有助于更准确地制定商超的销售策略，提高销售效益。
2. 适用于时间序列数据：采用LSTM（LSTM）结构的模型适用于处理时间序列数据，能够捕捉销售数据中的时间趋势和周期性，提高了模型的预测能力，有助于应对季节性需求波动。
3. 灵活性：遗传算法（GA）用于优化定价策略和损耗率，具有一定的灵活性，可以根据不同的商超需求和约束条件进行调整，满足不同场景下的优化需求。
4. 可解释性：模型的输出包括单品定价策略和损耗率，这些参数具有直观的商业含义，商超可以根据模型的结果做出决策，增加了决策的可解释性。
5. 非负销售量约束：模型中包括销售量的非负约束，确保了生成的定价策略和损耗率不会导致负销售量，更符合实际销售情况。

10.2模型缺点：

1. 数据需求：模型对大量的历史销售数据和定价数据的需求较高，如果数据质量不佳或不充分，模型的性能可能会受到限制，需要解决数据获取和质量问题。
2. 计算复杂性：遗传算法作为一种启发式算法，可能需要较多的计算资源和时间来搜索最优解，尤其是在参数空间较大的情况下，需要注意计算效率。
3. 模型复杂性：LSTM模型的复杂性相对较高，需要调整许多超参数，包括隐藏层的大小、学习率等，因此模型的调优可能需要一定的经验和时间，需要仔细优化模型参数。
4. 局部最优解：尽管遗传算法具有一定的全局搜索能力，但仍然可能陷入局部最优解，特别是在参数空间复杂且存在多个峰值的情况下，需要谨慎处理优化问题。
5. 依赖数据质量：模型对输入数据的质量和准确性敏感，如果存在噪声或异常值，可能会影响模型的性能和稳定性，需要进行数据清洗和异常值处理。

综合而言，该模型在处理销售量与定价之间的关系问题上具有一定的优势，但也存在一些挑战。在实际应用中，需要根据商超的具体情况和可用数据来权衡这些优缺点，以确定是否适合采用此模型来优化销售策略，同时需要不断改进和优化模型以提高性能。

1. 参考文献

[1]徐维超.相关系数研究综述[J]. 广东工业大学学报,2012,29(3):12-17

[2]姜启源.数学模型（第五版）[M].北京：高等教育出版社，2018.

[3]陈亮,王震,王刚.深度学习框架下LSTM网络在短期电力负荷预测中的应用[J].电力信息与通信技术,2017(5):8-11.

[4]柯苗,黄华国.基于LSTM神经网络的电商商品销售预测方法[J].福建师大福清分校学报,2020(05):83-89.

[5] Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2019). Multivariate Data Analysis (8th ed.). Cengage Learning.

附录

文件一：数据处理结果.xlsx

文件二：不同品类之间的关联程度.xlsx

文件三：不同单品之间的关联程度,xlsx

文件四：蔬菜品类销量与定价关系.xlsx

文件五：各蔬菜品类未来一周(2023 年 7 月 1-7 日)的日补货总量和定价策略.xlsx

文件六：2023年6月24日至30日的可售品种.xlsx

文件七：问题三结果.xlsx

模型一: 品类销量与定价预测模型sales\_prediction\_model\_with\_category.h5

模型二：单品销量与定价预测模型classification\_model.h5

代码一：基于问题一关联度模型使用python对不同单品之间的关联度求解

import pandas as pd

import numpy as np

import scipy.stats as stats

# 读取数据预处理的Excel文件

attachment2 = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\附件2.xlsx")

# 数据预处理：将销售日期转换为日期格式

attachment2["销售日期"] = pd.to\_datetime(attachment2["销售日期"])

# 先确保销量列的数据都是字符串

attachment2["销量(千克)"] = attachment2["销量(千克)"].astype(str)

# 移除销量中的空格，然后将其转换为浮点数

attachment2["销量(千克)"] = attachment2["销量(千克)"].str.replace(" ", "").astype(float)

# 初始化一个空的相关系数列表

correlation\_list = []

# 针对每对单品（A和B），执行以下步骤：

unique\_products = attachment2["单品编码"].unique()

for i, product\_A in enumerate(unique\_products):

for product\_B in unique\_products[i + 1:]: # 避免重复计算

# 获取单品A和B的销售数据

sales\_A = attachment2.loc[attachment2["单品编码"] == product\_A][["销售日期", "销量(千克)"]]

sales\_B = attachment2.loc[attachment2["单品编码"] == product\_B][["销售日期", "销量(千克)"]]

# 合并单品A和B的销售数据，基于销售日期进行匹配

merged\_sales = sales\_A.merge(sales\_B, on="销售日期", suffixes=("\_A", "\_B"), how="inner")

# 检查销售数据的标准差是否足够大，如果不够大就跳过计算

if merged\_sales["销量(千克)\_A"].std() > 0.001 and merged\_sales["销量(千克)\_B"].std() > 0.001:

# 执行多元斯皮尔曼秩相关系数计算

rho, \_ = stats.spearmanr(merged\_sales["销量(千克)\_A"], merged\_sales["销量(千克)\_B"])

# 将计算得到的相关系数添加到相关系数列表中

correlation\_list.append({

"单品A": product\_A,

"单品B": product\_B,

"多元斯皮尔曼秩相关系数": rho

})

# 将相关系数列表转化为DataFrame格式

correlation\_df = pd.DataFrame(correlation\_list)

# 根据多元斯皮尔曼秩相关系数对单品之间的关联度进行排序

sorted\_correlation\_df = correlation\_df.sort\_values(by="多元斯皮尔曼秩相关系数", ascending=False)

# 保存结果为Excel文件

sorted\_correlation\_df.to\_excel("不同单品之间的关联程度.xlsx", index=False)

代码二：基于问题一关联度模型使用python对不同品类之间的关联度求解

import pandas as pd

import numpy as np

import scipy.stats as stats

# 读取数据预处理的Excel文件

attachment2 = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\附件2.xlsx")

# 读取附件一的Excel文件，包含单品编码和分类编码

attachment1 = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\CUMCM2023Problems\C题\附件1.xlsx")

# 创建单品到品类的映射表

product\_category\_mapping = dict(zip(attachment1["单品编码"], attachment1["分类名称"]))

# 数据预处理：将销售日期转换为日期格式

attachment2["销售日期"] = pd.to\_datetime(attachment2["销售日期"])

# 先确保销量列的数据都是字符串

attachment2["销量(千克)"] = attachment2["销量(千克)"].astype(str)

# 移除销量中的空格，然后将其转换为浮点数

attachment2["销量(千克)"] = attachment2["销量(千克)"].str.replace(" ", "").astype(float)

# 初始化一个空的相关系数列表

correlation\_list = []

# 针对每对单品（A和B），执行以下步骤：

unique\_products = attachment2["单品编码"].unique()

for i, product\_A in enumerate(unique\_products):

for product\_B in unique\_products[i + 1:]: # 避免重复计算

# 获取单品A和B的销售数据

sales\_A = attachment2.loc[attachment2["单品编码"] == product\_A][["销售日期", "销量(千克)"]]

sales\_B = attachment2.loc[attachment2["单品编码"] == product\_B][["销售日期", "销量(千克)"]]

# 合并单品A和B的销售数据，基于销售日期进行匹配

merged\_sales = sales\_A.merge(sales\_B, on="销售日期", suffixes=("\_A", "\_B"), how="inner")

# 检查销售数据的标准差是否足够大，如果不够大就跳过计算

if merged\_sales["销量(千克)\_A"].std() > 0.001 and merged\_sales["销量(千克)\_B"].std() > 0.001:

# 执行多元斯皮尔曼秩相关系数计算

rho, \_ = stats.spearmanr(merged\_sales["销量(千克)\_A"], merged\_sales["销量(千克)\_B"])

# 将计算得到的相关系数添加到相关系数列表中

correlation\_list.append({

"单品A": product\_A,

"单品B": product\_B,

"多元斯皮尔曼秩相关系数": rho

})

# 将相关系数列表转化为DataFrame格式

correlation\_df = pd.DataFrame(correlation\_list)

# 根据多元斯皮尔曼秩相关系数对单品之间的关联度进行排序

sorted\_correlation\_df = correlation\_df.sort\_values(by="多元斯皮尔曼秩相关系数", ascending=False)

# 步骤 2: 创建品类关联度字典

category\_correlations = {}

# 步骤 3: 遍历单品关联度数据框

for index, row in sorted\_correlation\_df.iterrows():

product\_A = row["单品A"]

product\_B = row["单品B"]

correlation = row["多元斯皮尔曼秩相关系数"]

# 步骤 4: 获取单品 A 和 B 所属的品类

category\_A = product\_category\_mapping.get(product\_A, None)

category\_B = product\_category\_mapping.get(product\_B, None)

# 步骤 5: 如果单品 A 和 B 都有所属品类，计算加权平均关联度

if category\_A and category\_B:

# 步骤 6: 计算权重（可以根据销售量等因素进行权重计算）

weight\_A = 1.0 # 可以根据需要进行权重计算

weight\_B = 1.0 # 可以根据需要进行权重计算

weighted\_correlation = correlation \* (weight\_A + weight\_B) / 2.0

# 步骤 7: 更新品类关联度字典

if category\_A not in category\_correlations:

category\_correlations[category\_A] = {}

if category\_B not in category\_correlations:

category\_correlations[category\_B] = {}

# 步骤 8: 记录品类 A 和 B 之间的关联度

category\_correlations[category\_A][category\_B] = weighted\_correlation

category\_correlations[category\_B][category\_A] = weighted\_correlation

# 步骤 9: 将品类关联度字典保存为Excel文件

category\_correlations\_df = pd.DataFrame(category\_correlations)

category\_correlations\_df.to\_excel("品类关联度.xlsx", index=True)

代码三: 建立品类售量与定价的LSTM回归预测模型

import pandas as pd

import numpy as np

import tensorflow as tf

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Input, Embedding, Flatten, Concatenate

from tensorflow.keras.optimizers import Adam

# 读取数据

data = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\附录\数据处理结果.xlsx", usecols=["销售日期", "销量(千克)", "分类名称"])

data["销售日期"] = pd.to\_datetime(data["销售日期"])

# 使用LabelEncoder对分类名称进行编码

le = LabelEncoder()

data["分类编码"] = le.fit\_transform(data["分类名称"])

# 标准化销量数据

scaler = StandardScaler()

data["销量(千克)"] = scaler.fit\_transform(data[["销量(千克)"]])

# 数据准备

X\_sales\_date = data["销售日期"].values.astype(np.int64) // 10 \*\* 9 # 转换为Unix时间戳

X\_category = data["分类编码"].values

y = data["销量(千克)"].values

X\_train\_sales\_date, X\_test\_sales\_date, X\_train\_category, X\_test\_category, y\_train, y\_test = train\_test\_split(

X\_sales\_date, X\_category, y, test\_size=0.2, random\_state=42)

# 构建LSTM模型

input\_sales\_date = Input(shape=(1,), name="sales\_date\_input")

input\_category = Input(shape=(1,), name="category\_input")

embedding\_layer = Embedding(input\_dim=len(np.unique(X\_category)), output\_dim=10)(input\_category)

flatten\_layer = Flatten()(embedding\_layer)

concat\_layer = Concatenate()([input\_sales\_date, flatten\_layer])

dense\_layer = Dense(64, activation='relu')(concat\_layer)

output\_layer = Dense(1)(dense\_layer)

model = tf.keras.Model(inputs=[input\_sales\_date, input\_category], outputs=output\_layer)

# 更改优化器为Adam，尝试不同的学习率

model.compile(loss='mean\_squared\_error', optimizer=Adam(lr=0.01)) # 尝试不同的学习率值

# 模型训练

model.fit([X\_train\_sales\_date, X\_train\_category], y\_train, epochs=10, batch\_size=64) # 增加批量大小

# 模型性能评估

y\_pred = model.predict([X\_test\_sales\_date, X\_test\_category])

mse = mean\_squared\_error(y\_test, y\_pred)

print("均方误差 (MSE):", mse)

# 保存模型

model.save("sales\_prediction\_model\_with\_category.h5")

代码四：基于售量与定价的回归预测模型分析各蔬菜品类的销售总量与成本加成定价的关系

import pandas as pd

import numpy as np

import tensorflow as tf

from sklearn.preprocessing import StandardScaler, LabelEncoder

from tensorflow.keras.models import load\_model, Model

from tensorflow.keras.layers import Input, Dense

import matplotlib.pyplot as plt

from matplotlib.font\_manager import FontProperties

# 设置字体

plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用中文字体（SimHei）来显示中文

# 读取数据

data = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\附录\数据处理结果.xlsx", usecols=["销售日期", "销量(千克)", "分类名称", "销售单价(元/千克)"])

data["销售日期"] = pd.to\_datetime(data["销售日期"])

# 使用LabelEncoder对分类名称进行编码

le = LabelEncoder()

data["分类编码"] = le.fit\_transform(data["分类名称"])

# 标准化销量和定价数据

scaler = StandardScaler()

data[["销量(千克)", "销售单价(元/千克)"]] = scaler.fit\_transform(data[["销量(千克)", "销售单价(元/千克)"]])

# 加载已训练的模型

loaded\_model = load\_model("sales\_prediction\_model\_with\_category.h5")

# 用户输入分类名称和销售单价

user\_categories = ["辣椒类", "茄类", "花叶类", "食用菌", "花菜类", "水生根茎类"] # 请替换为用户输入的分类名称列表

user\_cost\_markup\_range = list(range(1, 16)) # 请替换为用户输入的销售单价范围

# 创建DataFrame用于保存结果

results\_df = pd.DataFrame(columns=user\_categories, index=user\_cost\_markup\_range)

# 构建预测输入

for category in user\_categories:

user\_category\_encoded = le.transform([category])

X\_user\_category = np.array(user\_category\_encoded).reshape(1, -1)

for user\_cost\_markup in user\_cost\_markup\_range:

X\_user\_cost\_markup = np.array([[user\_cost\_markup]]) # 只提供销售单价

# 进行销量预测

predicted\_sales = loaded\_model.predict([X\_user\_cost\_markup, X\_user\_category])

# 构建反标准化模型

input\_layer = Input(shape=(1,))

output\_layer = Dense(2)(input\_layer) # 2表示输出2个神经元，匹配scaler.inverse\_transform期望的形状

inverse\_scaler\_model = Model(inputs=input\_layer, outputs=output\_layer)

# 对预测结果进行反标准化

predicted\_sales = inverse\_scaler\_model.predict(predicted\_sales)

# 保存结果

results\_df.loc[user\_cost\_markup, category] = predicted\_sales[0][0]

# 输出结果为Excel文件

results\_df.to\_excel("predicted\_sales\_results.xlsx")

# 绘制散点折线图

font\_path = "path/to/your/font.ttf" # 设置字体文件路径，替换为您的字体文件路径

font\_prop = FontProperties(fname=font\_path)

plt.figure(figsize=(10, 6))

for category in user\_categories:

plt.plot(results\_df.index, results\_df[category], marker='o', label=category)

plt.xlabel("定价", fontproperties=font\_prop) # 指定字体属性

plt.ylabel("销量", fontproperties=font\_prop) # 指定字体属性

plt.title("不同品类的销量随定价的变化", fontproperties=font\_prop) # 指定字体属性

plt.legend(prop=font\_prop) # 指定字体属性

plt.grid(True)

# 保存图表为图片文件

plt.savefig("sales\_vs\_price.png")

# 显示图表

plt.show()

# 输出预测结果

print("预测结果已保存为 predicted\_sales\_results.xlsx 文件。")

print("散点折线图已保存为 sales\_vs\_price.png 文件。")

代码五：对各品类的成本的计算

import pandas as pd

# 读取附件三数据

data\_cost = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\CUMCM2023Problems\C题\附件3.xlsx", usecols=["单品编码", "批发价格(元/千克)"])

# 读取附件一数据

data\_categories = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\CUMCM2023Problems\C题\附件1.xlsx", usecols=["单品编码", "分类名称"])

# 合并两个数据集

merged\_data = pd.merge(data\_cost, data\_categories, on="单品编码")

# 计算不同品类的平均成本

average\_cost\_by\_category = merged\_data.groupby("分类名称")["批发价格(元/千克)"].mean()

# 打印结果

print(average\_cost\_by\_category)

代码六：对各品类的损耗率的计算

import pandas as pd

# 读取附件一数据

data\_categories = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\CUMCM2023Problems\C题\附件1.xlsx", usecols=["单品编码", "分类名称"])

# 读取附件四数据，指定"A"列和"C"列作为需要的列

data\_loss\_rate = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\附件4.xlsx", usecols=["A", "C"])

# 重命名列名为 "单品编码" 和 "损耗率(%)"

data\_loss\_rate.columns = ["单品编码", "损耗率(%)"]

# 合并两个数据集

merged\_data = pd.merge(data\_categories, data\_loss\_rate, on="单品编码")

# 计算每个品类的总销售量和总损耗率

category\_loss\_rate = merged\_data.groupby("分类名称")["损耗率(%)"].mean()

# 打印结果

print(category\_loss\_rate)

代码七基于遗传算法对收益最大化优化模型的求解

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras.models import load\_model

from scipy.optimize import minimize

from sklearn.preprocessing import StandardScaler, LabelEncoder

# 读取数据

data = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\附录\数据处理结果.xlsx",

usecols=["销售日期", "销量(千克)", "分类名称"])

data["销售日期"] = pd.to\_datetime(data["销售日期"])

# 使用LabelEncoder对分类名称进行编码

le = LabelEncoder()

data["分类编码"] = le.fit\_transform(data["分类名称"])

# 标准化销量数据

scaler = StandardScaler()

data["销量(千克)"] = scaler.fit\_transform(data[["销量(千克)"]])

# 数据准备

X\_sales\_date = data["销售日期"].values.astype(np.int64) // 10 \*\* 9 # 转换为Unix时间戳

X\_category = data["分类编码"].values

y = data["销量(千克)"].values

# 加载训练好的LSTM模型

trained\_model = load\_model("sales\_prediction\_model\_with\_category.h5")

# 定义损失率和批发价格

category\_loss\_constraints = {

"水生根茎类": 9.46,

"花叶类": 9.99,

"花菜类": 9.57,

"茄类": 9.52,

"辣椒类": 9.44,

"食用菌": 9.55

}

wholesale\_prices = {

"水生根茎类": 8.085504,

"花叶类": 4.446458,

"花菜类": 5.771536,

"茄类": 5.689737,

"辣椒类": 7.692309,

"食用菌": 6.146643

}

# 定义目标函数

def objective\_function(prices):

# 计算总收益

total\_revenue = 0

total\_loss = 0

# 批处理优化

batch\_size = 10000

for start in range(0, len(X\_sales\_date), batch\_size):

end = start + batch\_size

batch\_X\_sales\_date = X\_sales\_date[start:end]

batch\_X\_category = X\_category[start:end]

sales\_predictions = trained\_model.predict([batch\_X\_sales\_date, batch\_X\_category])

total\_revenue += np.sum(sales\_predictions \* prices)

# 计算总损耗

total\_loss += np.sum([category\_loss\_constraints[le.inverse\_transform([category])[0]] \* np.sum(sales\_predictions[batch\_X\_category == category]) for category in np.unique(batch\_X\_category)])

# 最大化总收益，同时满足损耗约束

return -(total\_revenue - total\_loss)

# 构建初始价格向量

initial\_prices = np.array([wholesale\_prices[le.inverse\_transform([category])[0]] for category in X\_category])

# 定义约束条件列表

def category\_loss\_constraint(prices, category):

# 批处理优化

total\_loss = 0

batch\_size = 10000

for start in range(0, len(X\_sales\_date), batch\_size):

end = start + batch\_size

batch\_X\_sales\_date = X\_sales\_date[start:end]

batch\_X\_category = X\_category[start:end]

# 计算该品类的销售量预测

batch\_sales\_predictions = trained\_model.predict([batch\_X\_sales\_date, np.array([category])])

# 计算该品类的总损耗

total\_loss += np.sum(category\_loss\_constraints[le.inverse\_transform([category])[0]] \* np.sum(batch\_sales\_predictions) - np.sum(batch\_sales\_predictions))

return total\_loss

constraints = [{'type': 'ineq', 'fun': category\_loss\_constraint, 'args': (category,)} for category in np.unique(X\_category)]

# 最大化总收益，同时满足约束条件

result = minimize(objective\_function, initial\_prices, constraints=constraints, method='SLSQP')

# 输出最优的定价策略

optimal\_prices = result.x

# 输出结果

for category, price in zip(np.unique(X\_category), optimal\_prices):

print(f"Category: {le.inverse\_transform([category])[0]}, Optimal Price: {price}")

代码八 基于问题二的基础上LSTM方法对单品售量与定价的LSTM回归预测模型的建立

import pandas as pd

import numpy as np

import tensorflow as tf

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

# 读取Excel数据

data = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\附录\数据处理结果.xlsx")

# 选择需要的特征列

selected\_features = ["销售日期", "销量(千克)", "销售单价(元/千克)", "单品编码"]

data = data[selected\_features]

# 将销售日期列转换为日期时间类型

data["销售日期"] = pd.to\_datetime(data["销售日期"])

# 使用LabelEncoder对单品编码进行编码

le = LabelEncoder()

data["单品编码"] = le.fit\_transform(data["单品编码"])

# 选择用于模型训练的特征和目标列

features = ["销量(千克)", "销售单价(元/千克)", "销售日期", "单品编码"]

target = ["销量(千克)"]

# 标准化特征数据

scaler = StandardScaler()

data[features] = scaler.fit\_transform(data[features])

# 构建多个模型，每个单品对应一个模型

unique\_items = data["单品编码"].unique()

models = {}

# 划分训练集和测试集，构建并训练多个模型

for item\_code in unique\_items:

item\_data = data[data["单品编码"] == item\_code]

X = item\_data[features].values

y = item\_data[target].values

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 初始化模型

input\_dim = X.shape[1] # 输入特征的维度

hidden\_units = 64 # 隐藏层的神经元数量

output\_dim = 1 # 输出维度，即销售量的预测

model = tf.keras.Sequential([

tf.keras.layers.Input(shape=(input\_dim,)), # 输入层

tf.keras.layers.Dense(hidden\_units, activation='relu'), # 隐藏层

tf.keras.layers.Dense(output\_dim) # 输出层

])

# 定义损失函数

def custom\_loss(y\_true, y\_pred):

mse = tf.keras.losses.MeanSquaredError()

return mse(y\_true, y\_pred)

# 选择学习率

learning\_rate = 0.001

optimizer = tf.keras.optimizers.SGD(learning\_rate=learning\_rate)

# 编译模型

model.compile(optimizer=optimizer, loss=custom\_loss)

# 迭代训练

num\_epochs = 100 # 训练轮数

batch\_size = 32 # 小批量样本大小

for epoch in range(num\_epochs):

for i in range(0, len(X\_train), batch\_size):

# 获取一个小批量样本

batch\_X = X\_train[i:i + batch\_size]

batch\_y = y\_train[i:i + batch\_size]

# 计算损失并更新参数

with tf.GradientTape() as tape:

predictions = model(batch\_X)

loss = custom\_loss(batch\_y, predictions)

gradients = tape.gradient(loss, model.trainable\_variables)

optimizer.apply\_gradients(zip(gradients, model.trainable\_variables))

# 计算测试集上的损失

test\_predictions = model(X\_test)

test\_loss = custom\_loss(y\_test, test\_predictions)

# 每个epoch结束后，打印训练损失和测试损失

print(f"Item Code: {item\_code}, Epoch {epoch + 1}/{num\_epochs}, Test Loss: {test\_loss.numpy()}")

# 保存训练好的模型

models[item\_code] = model

# 模型训练完成，可以用于预测销售量

代码九 根据遗传算法对单品收益最大化优化模型的求解

import numpy as np

import random

import tensorflow as tf

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from geneticalgorithm import geneticalgorithm as ga # 需要安装 geneticalgorithm 库

# 读取Excel数据

data = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\附录\数据处理结果.xlsx")

# 选择需要的特征列

selected\_features = ["销售日期", "销量(千克)", "销售单价(元/千克)", "单品编码", "是否打折销售"]

data = data[selected\_features]

# 将销售日期列转换为日期时间类型

data["销售日期"] = pd.to\_datetime(data["销售日期"])

# 使用LabelEncoder对单品编码进行编码

le = LabelEncoder()

data["单品编码"] = le.fit\_transform(data["单品编码"])

# 选择用于模型训练的特征和目标列

features = ["销售单价(元/千克)", "销售日期", "单品编码", "是否打折销售"]

target = ["销量(千克)"]

# 标准化特征数据

scaler = StandardScaler()

data[features] = scaler.fit\_transform(data[features])

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[features], data[target], test\_size=0.2, random\_state=42)

# 构建回归模型（与上一段代码相同）

model = tf.keras.Sequential([

tf.keras.layers.Input(shape=(X\_train.shape[1],)), # 输入层

tf.keras.layers.LSTM(64, activation='relu'), # LSTM隐藏层

tf.keras.layers.Dense(1) # 输出层

])

# 定义损失函数

def custom\_loss(y\_true, y\_pred):

mse = tf.keras.losses.MeanSquaredError()

return mse(y\_true, y\_pred)

# 选择学习率

learning\_rate = 0.001

optimizer = tf.keras.optimizers.SGD(learning\_rate=learning\_rate)

# 编译模型

model.compile(optimizer=optimizer, loss=custom\_loss)

# 训练模型

num\_epochs = 100 # 训练轮数

batch\_size = 32 # 小批量样本大小

for epoch in range(num\_epochs):

for i in range(0, len(X\_train), batch\_size):

batch\_X = X\_train[i:i + batch\_size]

batch\_y = y\_train[i:i + batch\_size]

with tf.GradientTape() as tape:

predictions = model(batch\_X)

loss = custom\_loss(batch\_y, predictions)

gradients = tape.gradient(loss, model.trainable\_variables)

optimizer.apply\_gradients(zip(gradients, model.trainable\_variables))

# 每个epoch结束后，打印训练损失

print(f"Epoch {epoch + 1}/{num\_epochs}, Train Loss: {loss.numpy()}")

# 使用遗传算法进行参数优化

def fitness(params):

# 解码参数

lstm\_units, learning\_rate = params

# 构建模型

genetic\_model = tf.keras.Sequential([

tf.keras.layers.Input(shape=(X\_train.shape[1],)),

tf.keras.layers.LSTM(lstm\_units, activation='relu'),

tf.keras.layers.Dense(1)

])

# 编译模型

genetic\_model.compile(optimizer=tf.keras.optimizers.SGD(learning\_rate=learning\_rate), loss=custom\_loss)

# 训练模型

num\_epochs = 100 # 训练轮数

batch\_size = 32 # 小批量样本大小

for epoch in range(num\_epochs):

for i in range(0, len(X\_train), batch\_size):

batch\_X = X\_train[i:i + batch\_size]

batch\_y = y\_train[i:i + batch\_size]

with tf.GradientTape() as tape:

predictions = genetic\_model(batch\_X)

loss = custom\_loss(batch\_y, predictions)

gradients = tape.gradient(loss, genetic\_model.trainable\_variables)

optimizer.apply\_gradients(zip(gradients, genetic\_model.trainable\_variables))

# 预测销售量

y\_pred = genetic\_model.predict(X\_test)

# 计算模型性能指标

mse = mean\_squared\_error(y\_test, y\_pred)

return mse

# 定义参数范围和遗传算法参数

varbound = np.array([[32, 128], [0.0001, 0.01]]) # 参数范围

algorithm\_param = {'max\_num\_iteration': 100, 'population\_size': 10} # 遗传算法参数

# 使用遗传算法求解最佳参数

model\_genetic = ga(function=fitness, dimension=2, variable\_type='real', variable\_boundaries=varbound,

algorithm\_parameters=algorithm\_param)

# 运行遗传算法

model\_genetic.run()

# 获取最佳参数

best\_params = model\_genetic.output\_dict['variable']

# 输出最佳参数

print("Best Parameters (LSTM Units, Learning Rate):", best\_params)

代码十模型评估代码

import pandas as pd

import numpy as np

import tensorflow as tf

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# 读取Excel数据

data = pd.read\_excel(r"C:\Users\DIUDIUDIU\Desktop\附录\数据处理结果.xlsx")

# 选择需要的特征列

selected\_features = ["销售日期", "销量(千克)", "销售单价(元/千克)", "单品编码"]

data = data[selected\_features]

# 将销售日期列转换为日期时间类型

data["销售日期"] = pd.to\_datetime(data["销售日期"])

# 使用LabelEncoder对单品编码进行编码

le = LabelEncoder()

data["单品编码"] = le.fit\_transform(data["单品编码"])

# 选择用于模型训练的特征和目标列

features = ["销量(千克)", "销售单价(元/千克)", "销售日期", "单品编码"]

target = ["销量(千克)"]

# 标准化特征数据

scaler = StandardScaler()

data[features] = scaler.fit\_transform(data[features])

# 加载预训练模型

pretrained\_model = tf.keras.models.load\_model('classification\_model.h5')

# 选择测试集数据

X\_test = data[features].values

y\_test = data[target].values

# 进行模型预测

y\_pred = pretrained\_model.predict(X\_test)

# 计算模型性能指标

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# 打印性能指标

print(f"Mean Squared Error (MSE): {mse}")

print(f"Root Mean Squared Error (RMSE): {rmse}")

print(f"Mean Absolute Error (MAE): {mae}")

print(f"R-squared (R2): {r2}")