同濟大學

TONGJI UNIVERSITY

机器学习线性回归实验报告

学号	姓名
2150772	张哲源
2152568	孙治弘
2153771	胡语诺
2150792	牛禄阳
授课老师	李洁老师

一、 课题综述

1.1 课题说明

研究课题：北京二手房的房价预测

小组成员分工：

张哲源：统筹并协调组员进度，获取数据集，数据预处理。

牛禄阳：数据预处理与数据分割。

孙治弘：模型的构建与训练。

胡语诺：模型的构建与训练。

小组协作：模型训练结果与数据处理之间的交互，比如：特征选择、处理缺失数据等迭代过程；模型与数据的优化过程以及模型可视化结果。

1.2 课题目标

本课题的主要任务是使用线性回归模型来建立一个房价预测模型，该模型能够根据房屋的各种特征（面积、卧室数量、浴室数量等）来预测房屋的销售价格，为房地产市场或投资决策提供有用的工具和见解。

数据获取与探索：加载房价预测数据集，了解数据的特征和分布
数据预处理：进行数据清洗、处理缺失值、处理异常值等预处理步骤
特征选择与工程：分析数据特征，选择对房价预测最具影响力的特征
数据分割：将数据集分为训练集和测试集，通常采用 8/2 的比例。
模型选择与训练：选择线性回归模型，使用训练集训练模型，拟合房价与特征的关系。
模型评估：使用均方根误差（RMSE）度量线性回归模型的性能以评估模型的拟合程度。
算法复现: 复现线性回归模型的算法,不使用标准库.
模型解释与可视化：解释模型中各特征对房价的影响程度，可视化模型的拟合效果，使结果易于理解和传达。
结果总结与报告：总结课题的主要发现，阐述所构建房价预测模型的性能和应用潜力。

1.3 课题数据集

本课题使用的数据集来自 Kaggle 竞赛“房价预测”（House Price Prediction），数据集包含有关不同房屋的特征和相应的销售价格。以下是数据集的详细信息：

数据来源： https://www.kaggle.com/datasets/ruiqurm/lianjia

数据描述：数据集包含训练集和测试集两部分，其中训练集用于模型的训练和验证，测试集用于模型的最终评估和预测。

训练集包含了多个特征，如房屋的楼层、房屋的面积、卧室数量、浴室数量、地理位置等。训练集将每个房屋的销售价格作为目标变量。测试集包含相同的特征，但不包含目标变量，我们的任务是根据这些特征预测测试集中房屋的销售价格。

数据规模：总数据集包含约 300000 个样本（房屋信息），每个样本有 27 个特征，包括数值型和分类型特征。测试集通过数据集划分得来,使用留出法

数据质量：数据集经过了处理和清洗，确保了数据的质量和一致性。缺失数据已经得到处理，异常值已经被纠正或删除。

二、实验设计

2.1 数据准备

实验数据集的获取是本课题的重要步骤之一，我们需要确保数据集的可用性、完整性和质量。以下是实验数据集的获取过程：

数据来源：我们从 Kaggle 竞赛“房价预测”（House Price Prediction）中获取了数据集。Kaggle 是一个著名的数据科学竞赛平台，提供了大量丰富的数据集供数据科学家和研究人员使用。

数据文件格式：数据文件的结构和内容。数据结构为 `CSV` 格式，

数据质量检查：在导入数据之前，我们进行了数据质量检查，包括查找缺失值、异常值或数据不一致性。

数据导入：一旦数据集经过初步检查和处理，我们使用 Python 编程语言中的数据科学库（如 Pandas）来导入数据。这使我们能够在 `Jupyter Notebook` 中轻松处理和分析数据。

2.2 数据预处理

2.2.1 数据清洗

- **处理缺失值：**检测并处理数据集中的缺失值。我们可以选择删除包含缺失值的样本，或者填充缺失值，具体取决于数据的分布和业务需求(如下图所示)

缺失值处理

```
In [168]: # 均值填充缺失
data['DOM'].fillna(data['DOM'].dropna().mean(0),inplace=True)
# 建筑类型众数填补缺失
data['buildingType'] = data['buildingType'].fillna(data['buildingTyp
# 用中间值填充
data['communityAverage'] = data['communityAverage'].fillna(data['com
#房间等特征值的缺失就当做没有，用0填充
data['subway'] = data['subway'].fillna(0)
data['fiveYearsProperty'] = data['fiveYearsProperty'].fillna(0)
data['elevator'] = data['elevator'].fillna(0)

In [169]: data.isnull().sum().sort_values(ascending =False) # 缺失值处理完毕
Out[169]: url                 0
id                  0
district            0
subway              0
fiveYearsProperty   0

In [167]: data.isnull().sum().sort_values(ascending =False) # 统计缺失
Out[167]: DOM                 157977
buildingType          2021
communityAverage       463
subway                  32
fiveYearsProperty       32
elevator                32
bathRoom                 0
district                 0
ladderRatio              0
buildingStructure        0
renovationCondition      0
constructionTime         0
floor                    0
url                      0
id                       0
drawingRoom              0

'url','id','Cid','price'])
```

```
def haversine(lat1, lon1, lat2, lon2):
    # 将经纬度转换为弧度
    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    # 差值
    dlon = lon2 - lon1
    dlat = lat2 - lat1

    # 使用Haversine公式计算距离
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    # 地球半径（平均值，单位：千米）
    r = 6371.0

    # 计算距离
    distance = r * c

    return distance
```

```
In [174]: tian_an_meng_lat = 39.908692
          tian_an_meng_lng = 116.397477
```

```
In [175]: data['dis_to_tianmeng'] = data.apply(lambda row: haversine(row['Lat'], row['Lng'], tia
```

- **特征提取** 将经纬度变为到城中心的距离,这里以距离天安门的距离为例。

- **多值变量的分割** 将多值变量分割存储为单独的列，以便在数据分析中更轻松地进行处理和可视化，能够更好地利用这些数据进行建模和分析。本题将变量 floor 分割为 floor_type、floor_number 等多个变量。

```
In [16]: data.floor
```

```
Out[16]: 0         高 26
         1         高 22
         2          中 4
         3         底 21
         4          中 6
                  ...
         318846     中 5
         318847    中 24
         318848     中 7
         318849     中 5
         318850    中 17
         Name: floor, Length: 318851, dtype: object
```

```
In [176]: data[['floor_type', 'floor_number']] = data['floor'].str.replace(r'\s+', ' ',regex=Tru
          data = data.drop(columns=['floor'])
```

```
In [177]: data['tradeTime'] = pd.to_datetime(data['tradeTime'], format='%Y/%m/%d')
```

```
In [178]: data['year'] = data['tradeTime'].dt.year
          data['month'] = data['tradeTime'].dt.month
          data['day'] = data['tradeTime'].dt.day
          data.drop(columns=['tradeTime'], inplace=True)
```

- **分类特征编码** 将分类特征进行独热编码或标签编码，以便模型能够处理非数值型数据。

In [187]:
```
# 将 '未知' 替换为 NaN
data['constructionTime'].replace('未知', np.nan, inplace=True)

# 将 'constructionTime' 列转换为数值类型
data['constructionTime'] = pd.to_numeric(data['constructionTime'], errors='coerce')

# 计算 'constructionTime' 列的平均值
mean_constructionTime = data['constructionTime'].mean()

# 使用平均值填充缺失值
data['constructionTime'].fillna(mean_constructionTime, inplace=True)
```

In [188]:
```
# 使用get_dummies将'floor_type'列转换为独热编码
data = pd.get_dummies(data, columns=['floor_type'], prefix=['floor_type'], drop_first='
data.info()
data = data.replace({False:0,True:1})
```

- **类型转换** 将原始数据中的某些特征或标签从一种数据类型转换为另一种，以便与模型兼容。本题将一些 object 类型的变量修改为 int、bool 等类型，使得其更适配于回归模型的处理。

2.2.3 标准化

```
# 对数值型列进行标准化
standardized_data = scaler_standard.fit_transform(numerical_columns)

# 创建DataFrame来保存标准化后的数据
standardized_df = pd.DataFrame(standardized_data, columns=numerical_columns.columns)

# 合并bool和int32类型的列
result_df = pd.concat([standardized_df, data[['floor_type_低', 'floor_type_底', 'floor_
result_df = result_df.dropna(subset=columns_to_check, how='any')
result_df.to_csv('./stand_data.csv', index=False, encoding='utf-8')
```

In [197]:
```
result_df = result_df.dropna(subset=columns_to_check, how='any')
result_df.isnull().sum().sort_values(ascending =False) # 检查缺失值
```

Out[197]: floor_type_高 32

2.2.4 数据分割

使用`留出法`划分数据集为 5:1 训练集和测试集

通过上述数据预处理步骤，确保了数据的质量、适合模型的特征工程，以及数据的合理分割。

2.3 模型搭建

使用 PyTorch 的 `nn.Linear` 定义了一个线性回归模型，该模型包含一个线性层，输入特征的维度由数据的特征数量决定。

2.3.1 损失函数：

使用均方误差损失（MSE Loss）作为模型的损失函数，用于衡量模型的预测值与真实标签之间的差异。

2.3.2 训练过程：

实现了训练函数，其中使用了 Adam 优化器来优化模型参数。在每个训练周期内，对数据进行迭代，计算损失，进行梯度反向传播并更新模型参数。训练损失被记录并用于监测模型的训练进度。

本实验中使用了一个简单的线性回归模型，它由一个线性层组成，负责拟合输入特征与目标变量之间的关系。

```
def get_net():
    net = nn.Sequential(nn.Linear(in_features, 1))
    return net
```

2.3.3 损失函数

模型的损失函数采用了均方误差（MSE）损失函数，用于度量模型预测值与真实值之间的差距。

```
loss = nn.MSELoss()
```

2.3.4 训练和交叉验证

使用给定的训练数据和测试数据进行模型训练和性能评估。自动执行多个训练周期（epochs）以迭代优化模型参数。使用 Adam 优化器来更新模型的权重和偏置。计算和记录训练数据和测试数据的 log RMSE 损失。

```
def train(net, train_features, train_labels, test_features, test_labels,
              num_epochs, learning_rate, weight_decay, batch_size):
    train_ls, test_ls = [], []
    train_losses = []
    train_iter = d2l.load_array((train_features, train_labels), batch_size)
    optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate,
    weight_decay=weight_decay)
    for epoch in range(num_epochs):
        for X, y in train_iter:
            optimizer.zero_grad()
            l = loss(net(X), y)
            l.backward()
            optimizer.step()
        train_loss = log_rmse(net, train_features, train_labels)
```

```
            train_losses.append(train_loss)
            train_ls.append(log_rmse(net, train_features, train_labels))
            if test_labels is not None:
                test_ls.append(log_rmse(net, test_features, test_labels))
return train_ls, test_ls
```

实验中采用了 K 折交叉验证来评估 1 模型性能。K 折交叉验证将数据集分为 K 个折叠，每次训练模型时使用 K-1 个折叠作为训练集，剩余的一个折叠作为验证集。将数据集分为 K 份，进行 K 次训练和验证，并计算训练和验证的平均性能指标。

```
def k_fold(k, X_train, y_train, num_epochs, learning_rate, weight_decay,
  batch_size):
    train_l_sum, valid_l_sum = 0, 0
    for i in range(k):
        data = get_k_fold_data(k, i, X_train, y_train)
        net = get_net()
        train_ls, valid_ls = train(net, *data, num_epochs, learning_rate,
          weight_decay, batch_size)
        train_l_sum += train_ls[-1]
        valid_l_sum += valid_ls[-1]
        if i == 0:
            d2l.plot(list(range(1, num_epochs + 1)), [train_ls, valid_ls],
                     xlabel='epoch', ylabel='rmse', xlim=[1, num_epochs],
                     legend=['train', 'valid'], yscale='log')
        print(f'fold {i + 1}, train log rmse {float(train_ls[-1]):f}, '
              f'valid log rmse {float(valid_ls[-1]):f}')
    return train_l_sum / k, valid_l_sum / k
```

实现了 K 折交叉验证过程，用于评估线性回归模型在不同数据子集上的性能，并返回模型的平均性能指标，以便进行模型选择和性能评估。

2.3.5 Adma 优化器

```
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate,
weight_decay=weight_decay)
```

能够在训练过程中自动调整学习率，通常效果较好。这种优化器可以有效加速模型的收敛，并且对超参数的敏感度相对较低.

模型中使用的主要参数意义：

1. `loss = nn.MSELoss()`：定义损失函数的参数。`nn.MSELoss()` 表示均方误差损失函数，用于测量模型的预测值与真实标签之间的差距。该损失函数在模型训练中被优化，目标是最小化均方误差。

2. `in_features = train_features.shape[1]`：表示输入特征的数量。它的值是训练数据集 `train_features` 的列数，用于确定线性回归模型的输入维度。在模型定义时，会根据输入特征的数量来配置线性层的输入维度。
3. `num_epochs`：表示训练的总周期数，决定模型在训练数据上迭代多少次。
4. `learning_rate`：学习率，用于控制模型参数更新的步长，影响优化算法的收敛速度。
5. `weight_decay`：权重衰减（L2 正则化）参数，用于控制模型的复杂度，避免过拟合。
6. `batch_size`：每个小批量训练的样本数量，决定了每次迭代更新模型参数时所使用的样本数量。

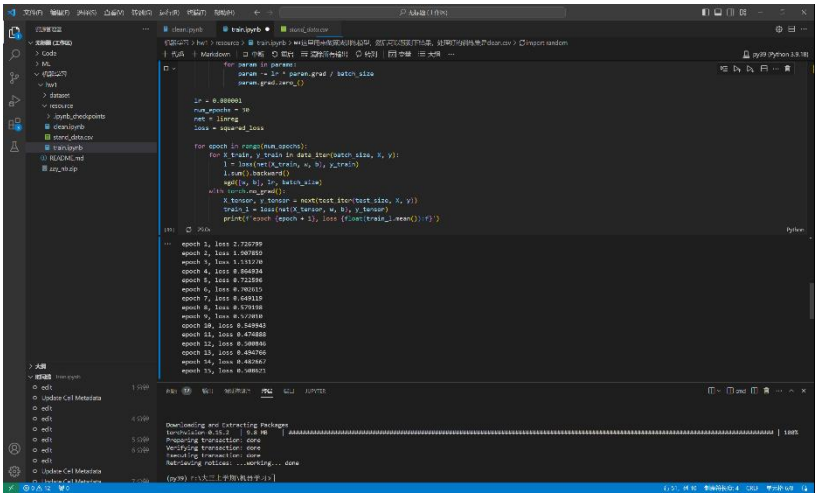
模型中使用到的主要变量意义：

1. `train_losses`：列表；用于存储每个训练周期的训练集上的对数均方根误差 (log RMSE)。它记录了模型在训练数据上的表现随着训练周期的变化。
2. `train_ls` 和 `test_ls`：列表；分别用于存储每个训练周期的训练集和测试集上的对数均方根误差 (log RMSE)。它们记录了模型在训练集和测试集上的性能随着训练的进行而变化。
3. `train_iter`：数据迭代器；用于将训练数据集划分为小批量进行训练。它根据 `batch_size` 将训练数据分成小批量，以便在每个训练周期内迭代使用。这有助于有效地进行随机梯度下降 (SGD) 训练。
4. `optimizer`：优化器

2.4 模型训练测试优化

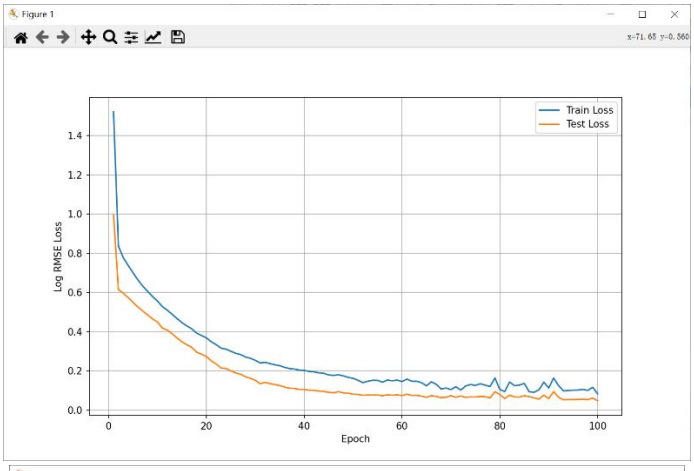
```
epoch 1, loss 0.533572
epoch 2, loss 0.508753
epoch 3, loss 2953.153809
epoch 4, loss 422102944.000000
epoch 5, loss 289358118912.000000
epoch 6, loss 6752173920616448.000000
epoch 7, loss 4707297438516463534080000.000000
epoch 8, loss 1042825340783124416351436800.000000
epoch 9, loss inf
epoch 10, loss inf
epoch 11, loss inf
epoch 12, loss inf
epoch 13, loss inf
epoch 14, loss inf
epoch 15, loss nan
epoch 16, loss nan
epoch 17, loss nan
epoch 18, loss nan
epoch 19, loss nan
epoch 20, loss nan
epoch 21, loss nan
epoch 22, loss nan
epoch 23, loss nan
epoch 24, loss nan
epoch 25, loss nan
epoch 26, loss nan
epoch 27, loss nan
```

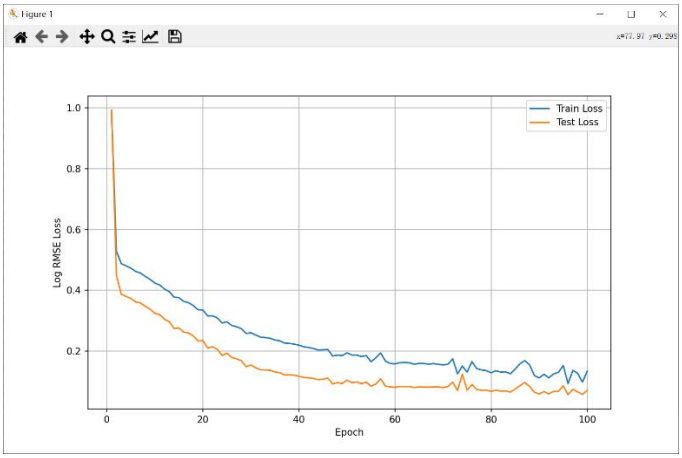
一开始，由于学习率设置过大，导致出现了梯度爆炸，损失函数变成了 nan，后来在不断的调试中，逐渐找到了合适的学习率，loss 也不再爆炸，如下图。



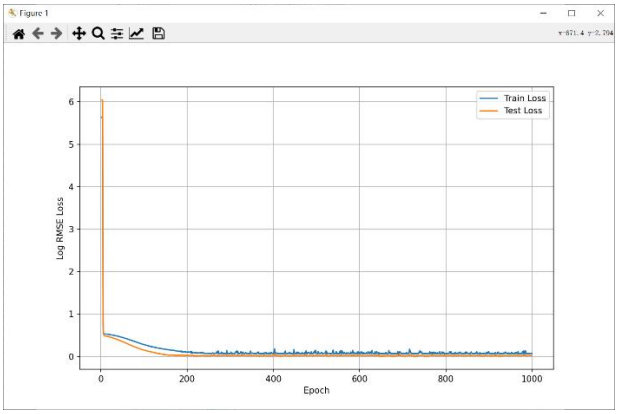
2.5 结果可视化

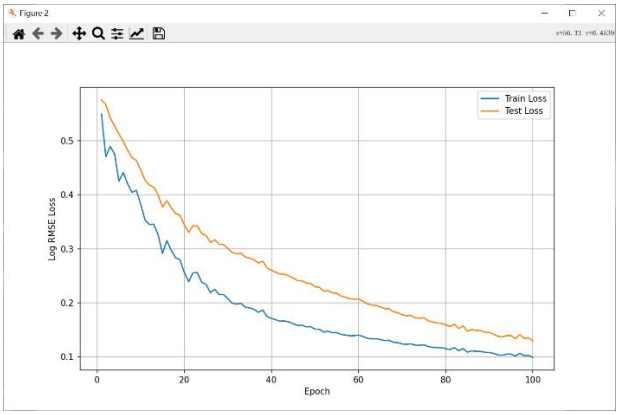
在学习率为 0.001，测试集大小为 5000，训练 100 轮，损失值如下。





在学习率为 0.001，测试集大小为 5000 时，训练 1000 轮时，损失值如下。





2.6 分析和优化

后续，我们可以继续使用一些优化方法实现更好的实验效果。SGD 和 Adam 是较为基本的优化器。SGD 在每个训练批次上更新模型参数，通过计算每个批次的梯度来进行参数更新，但通常需要精心调整学习率以获得最佳性能。

此外，我们还可以考虑使用 AdaGrad、Adadelta、L-BFGS、Rprop 和 Nadam 等方法进一步优化。

三、总结

首先我们挑选了适合的数据集，这是线性回归的关键，在数据预处理中我们对数据进行了各种方式的处理。数据集的质量和特征的选择对模型的性能有重大影响，便能够有效地拟合线性模型。

由于数据集较大，我们采用 K 折交叉验证进行处理，有助于避免过拟合，同时允许我们更好地了解模型在不同子集上的泛化能力。通过计算平均误差和标准差，我们能够更可靠地评估模型的性能。梯度下降优化：为了训练线性回归模型，我们选择了梯度下降作为优化算法。梯度下降的迭代过程允许模型逐渐调整参数以最小化损失函数。在实验中，我们调整了学习率和迭代次数，以找到最佳的参数设置。

线性回归是机器学习的基础，在本次实验中，我们对线性回归有了更深的理解，了解了机器学习的基本理念以及优化方法。实验最后我们也总结了实验结果，讨论了可能存在的改进点，为后续的学习打下了基础。