**同济大学计算机系**

**《机器学习》 实验报告**

**实验1 回归**

**徽标

描述已自动生成**

**学 号 1951705 1953493 1952219 1950455**

**姓 名**

**专 业 计算机科学与技术**

**授课老师 李洁**

目录

[1 实验概述 3](#_Toc99322893)

[1.1 实验目的 3](#_Toc99322894)

[1.2 实验内容 3](#_Toc99322895)

[2 实验方案设计 3](#_Toc99322896)

[2.1 总体设计思路与总体架构 3](#_Toc99322897)

[2.2 核心算法及基本原理 3](#_Toc99322898)

[2.3 模块设计：本实验的具体模块设计 7](#_Toc99322899)

[2.4 其他创新内容或优化算法 8](#_Toc99322900)

[2.4.1 手写线性模型梯度下降算法 8](#_Toc99322901)

[2.4.2 手写神经网络梯度下降算法 8](#_Toc99322902)

[2.4.3 Adam算法 9](#_Toc99322903)

[3 实验过程 10](#_Toc99322904)

[3.1 环境说明： 10](#_Toc99322905)

[3.1.1 操作系统 10](#_Toc99322906)

[3.1.2 开发语言 10](#_Toc99322907)

[3.1.3 开发环境及具体版本 10](#_Toc99322908)

[3.1.4 核心使用库 10](#_Toc99322909)

[3.2 源代码文件清单，主要函数清单 10](#_Toc99322910)

[3.3 实验过程 12](#_Toc99322911)

[3.3.1 数据预处理 12](#_Toc99322912)

[3.3.2 模型拟合 18](#_Toc99322913)

[3.3.3 模型比较 29](#_Toc99322914)

[3.4 实验结论 31](#_Toc99322915)

[4 总结 31](#_Toc99322916)

[4.1 实验中存在的问题及解决方案 31](#_Toc99322917)

[4.1.1 线性回归模型梯度下降法求解参数值时发生梯度爆炸 31](#_Toc99322918)

[4.1.2 无法通过最小二乘法求解Lasso模型的参数值 32](#_Toc99322919)

[4.1.3 神经网络反向传播时如何处理bias 32](#_Toc99322920)

[4.2 心得体会 33](#_Toc99322921)

[4.3 后续改进方向 34](#_Toc99322922)

[4.3.1 提升模型训练速度 34](#_Toc99322923)

[4.3.2 尝试使用更多模型拟合 34](#_Toc99322924)

[5 参考文献 34](#_Toc99322925)

[6 成员分工与自评 34](#_Toc99322926)

# 实验概述

## 实验目的

回归是一种用于金融、投资和其他学科的统计方法，它试图确定一个因变量（通常用 Y 表示）与一系列其他变量（称为自变量 X）之间关系的强度和特征。

本实验要求自行选择数据集，进行机器学习回归分析。在此基础上进行优化和进一步的讨论探究。

## 实验内容

实验大致分为以下步骤：

* + 数据清洗和预处理
  + 模型构建
  + 训练和测试
  + 绘制结果
  + 优化和查验

# 实验方案设计

## 总体设计思路与总体架构

1. 本小组选用混凝土抗压强度数据集（Concrete Compressive Strength Data Set）。

<https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength>

1. 对数据集进行清洗和预处理。
2. 使用常见的回归模型，简单地调用第三方库中提供的模块，对于该数据集进行机器学习回归分析，将结果可视化。
3. 基于sklearn选取多种模型进行尝试。
4. 手写实现回归算法模型。
5. 探讨模型优化细节，尝试更多的优化模型思路。
6. 根据实验结果完成讨论分析和实验报告

## 核心算法及基本原理

本实验涉及到的回归模型：

* + 线性回归

线性回归是一种线性模型，它假设输入变量 ( X) 和单个输出变量 ( y)之间存在线性关系。这种情况可以把数据画在图上显示出来，可以明显看到这种线性关系，表达式 ，表示预测值， 表示误差项，表示自变量的个数，是一个行向量， 表示每个自变量前面的系数，是一个行向量，表示一个与自变量无关的常数项。

一般来说，有两种情况：

1.单变量线性回归：它表示单个输入变量和单个的输出变量模型之间的关系。

2.多变量线性回归（也称为多元线性回归）：它对多个输入变量和单个输出变量之间的关系进行建模。

如何获得最佳拟合曲线这个问题可以使用最小二乘法完成。最小二乘法也是用于拟合回归线最常用的方法。对于观测数据，它通过最小化每个数据点到线的垂直偏差平方和来计算最佳拟合线。因为在相加时，偏差先平方，所以正值和负值没有抵消。

* + Ridge 回归

岭回归分析是一种用于存在多重共线性（自变量高度相关）数据的技术。在多重共线性情况下，尽管最小二乘法（OLS）对每个变量很公平，但它们的差异很大，使得观测值偏移并远离真实值。岭回归通过给回归估计上增加一个偏差度，来降低标准误差。

岭回归(Ridge Regression)是在平方误差的基础上增加正则项，岭回归通过收缩参数λ（lambda）解决多重共线性问题，见以下公式：

在这个公式中，有两个组成部分。第一个是最小二乘项，另一个是的倍，其中是相关系数。为了收缩参数把它添加到最小二乘项中以得到一个非常低的方差。是一个正则化方法，并且使用的是L2正则化。

* + LASSO 回归

它类似于岭回归，Lasso （Least Absolute Shrinkage and Selection Operator）也会惩罚回归系数的绝对值大小。此外，它能够减少变化程度并提高线性回归模型的精度。见下面的公式：

Lasso 回归与Ridge回归有一点不同，它使用的惩罚函数是绝对值，而不是平方。这导致惩罚值（约束估计的绝对值之和）使一些参数估计结果等于零。使用惩罚值越大，进一步估计会使得缩小值趋近于零。这将导致我们要从给定的n个变量中选择变量。这是一个正则化方法，使用的是L1正则化。

* + 弹性网络回归(ElasticNet)

ElasticNet是一种使用L1和L2先验作为正则化矩阵的线性回归模型。这种组合用于只有很少的权重非零的稀疏模型, 但是又能保持Ridge的正则化属性。同时，当多个特征和另一个特征相关的时候弹性网络非常有用,因为Lasso倾向于随机选择其中一个，而弹性网络更倾向于选择两个。它的目标函数为：

* + 决策树回归

决策树是一种基本的分类与回归方法。决策树由结点和有向边组成。结点有两种类型：内部结点和叶结点。内部结点表示一个特征或属性，叶结点表示一个类别或者某个值。

CART回归树采用启发式的方法对输入空间进行划分，选择第j个特征变量和它取的值s，作为切分特征和切分点，并采用mse作为选择最优划分点的标准，具体公式见下：

其中，

* + 随机森林回归

随机森林回归模型由多棵回归树构成，且森林中的每一棵决策树之间没有关联，模型的最终输出由森林中的每一棵决策树共同决定。

随机森林的随机性体现在两个方面：

1、样本的随机性，从训练集中随机抽取一定数量的样本，作为每颗回归树的根节点样本；

2、特征的随机性，在建立每颗回归树时，随机抽取一定数量的候选特征，从中选择最合适的特征作为分裂节点。

算法原理如下：

（a）从训练样本集S中随机的抽取m个样本点，得到一个新的S1…Sn个子训练集;

（b）用子训练集，训练一个CART回归树(决策树)，这里在训练的过程中，对每个节点的切分规则是先从所有特征中随机的选择k个特征，然后在从这k个特征中选择最优的切分点在做左右子树的划分。(这里的得到决策树都是二叉树)

（c）通过第二步，可以生成很多个CART回归树模型。

（d）每一个CART回归树最终的预测结果为该样本点所到叶节点的均值。

（e）随机森林最终的预测结果为所有CART回归树预测结果的均值。

一般来说，随机森林回归与决策树回归非常相似，它在数据集的各种子样本集上构建多个相互独立的基评估器（base estimator）（决策树），然后对其预测进行平均来决定结果，提高预测准确性和控制过拟合。这也就是袋装法（bagging）的过程。

随机森林的特点在于两个随机采样过程，随机森林对输入的数据要进行行(样本)、列(特征)的采样。对于样本采样，采用有放回的方式，也就是在采样得到的样本集合中，可能有重复的样本。一般情况下，回归树算法都一个重要的步骤——剪枝，但是在随机森林算法中 ，由于之前的两个随机采样的过程保证了随机性，所以就算不剪枝，也不会出现over-fitting。

* + 人工神经网络(多层感知机)

人工神经网络（Artificial Neural Network，ANN）在机器学习和认知科学领域，是一种模仿生物神经网络的结构和功能的数学模型或计算模型，用于对函数进行估计或近似。神经网络由大量的人工神经元联结进行计算。

其中，多层感知机（Multilayer Perceptron）是一种前向结构的人工神经网络，映射一组输入向量到一组输出向量。MLP可以被看作是一个有向图，由多个的节点层所组成，每一层都全连接到下一层。除了输入节点，每个节点都是一个带有非线性激活函数的神经元（或称处理单元）。反向传播算法的监督学习方法常被用来训练MLP。多层感知器遵循人类神经系统原理，学习并进行数据预测。它首先学习，然后使用权重存储数据，并使用算法来调整权重并减少训练过程中的偏差，即实际值和预测值之间的误差。多层感知机是单层感知机的推广，克服了单层感知机不能对线性不可分数据进行识别的弱点，主要优势在于其快速解决复杂问题的能力。

多层感知机由三层组成：输入层，隐藏层和输出层，除第一层为样本输入外，每一层的输入为上一层的激活值与本层权重的乘积和（类似前述线性回归模型）输入激活函数后的激活值。多层感知机的优势就在于这个激活函数，它大大增大的模型的复杂度。常见的激活函数有relu, leakyrelu, simoid, tanh等。

除第一层外，每一层中下一层的输入即

要更新神经网络内的参数值时，只需对每个样本，根据链式法则求出损失函数对于每个参数的偏导值，然后求出其平均值即为要更新的梯度。

该梯度乘以学习率即为神经网络参数值的更新值。

## 模块设计：本实验的具体模块设计

我们将实验分为以下几个模块，具体见下表：

|  |  |
| --- | --- |
| **模块名称** | **模块功能** |
| main | 主函数，调用其余模块以实现回归模型的构建与评估。 |
| Regression | 训练及评估：存放回归函数，采用K折交叉验证方法训练并评估传入的模型。 |
| Data\_preprocessing | 数据预处理：存放数据预处理相关函数，包括PCA、归一化等。 |
| LinearRegression | 线性回归：scikit-learn内置的线性回归模型 |
| LinearRegressionHandWrite | 手写线性回归：存放手写的线性回归模型及相关函数 |
| Lasso | LASSO回归模型：scikit-learn内置的LASSO回归模型 |
| LassoHandWrite | 手写LASSO模型：存放手写实现的LASSO回归模型及相关函数 |
| Ridge | 岭回归模型：scikit-learn内置的岭回归模型 |
| RidgeHandWrite | 手写岭回归模型：存放手写实现的岭回归模型及相关函数 |
| DecisionTreeRegressor | 决策树模型：scikit-learn内置的决策树模型 |
| DecisionTreeRegressorHandWrite | 手写决策树模型：存放手写实现的决策树模型及相关函数 |
| RandomForestRegressor | 随机森林模型：scikit内置的随机森林模型 |
| myRandomForest | 手写随机森林模型：存放手写实现的随机森林模型及相关函数 |
| MLPRegressor | 多层感知器模型：scikit-learn内置的多层感知器模型 |
| MLPHandWrite | 手写多层感知器模型：存放手写实现的多层感知器及相关函数 |

## 其他创新内容或优化算法

### 手写线性模型梯度下降算法

|  |
| --- |
| def gradient(Theta,X, y,reg\_const=0,L=0):#计算梯度,X已增广过,L为0则不正则化,否则为L1,L2      m, n = X.shape      theta=Theta      #theta=np.reshape(Theta,(n,1))      grad=np.zeros(n)      for t in range(m):#每个样本          x=X[t,:]          h=np.matmul(x,theta)          diff=h-y[t]          grad=grad+diff\*x        grad = grad / m #除以m      # 反向传播，额外计算正则化项(L1)      if(L==1):          grad = grad + reg\_const \* np.where(theta!=0,np.absolute(theta)/theta,0)#L1导数为绝对值除以自身      if(L==2):          grad = grad +reg\_const \* theta      return grad |

文本, 信件

描述已自动生成

如上代码，将所有参数视作一个矩阵考虑，则梯度下降法只需对遍历样本，然后分别求出每个样本对应的梯度，再求平均值即可。

对于LASSO和RIDGE算法，只需添加正则化项即可。

### 手写神经网络梯度下降算法

|  |
| --- |
| def gradient(Theta,network\_struct,reg\_const,X, y):#计算梯度      m, n = X.shape      cur, pos = 0, 0      theta\_list = []      for i in range(len(network\_struct) - 1):          pos = (network\_struct[i] + 1) \* network\_struct[i + 1] + pos          theta\_list.append(Theta[cur:pos].reshape((network\_struct[i + 1], network\_struct[i] + 1)))  # 下一层，前一层+1          cur = pos      Y = np.reshape(y, (m, 1))      grad\_list = [np.zeros(i.shape) for i in theta\_list]  # 梯度列表      for t in range(m):  # 做m次 m为训练集样本数          x = X[t, :]          A = x.reshape((1, n))  # 1\*(input+1)的矩阵          A\_list = [A]          Z\_list = [-1]  # 没有Z0          for i in range(len(theta\_list) - 1):              Z = np.dot(A, theta\_list[i].T)  # 转置 然后矩阵乘法得到1\*k              A = Activation\_Function(Z)              A = np.column\_stack((np.ones(1), A))  # 增广，方便下次运算              Z = np.column\_stack((np.ones(1), Z))  # 增广，方便下次运算              A\_list.append(A)              Z\_list.append(Z)          h = np.dot(A, theta\_list[-1].T)  # 最后一层 完成前向传播          diff = h - Y[t, :]  # 与对应样本求差          grad\_list[-1] = grad\_list[-1] + np.dot(diff, A\_list[-1])  # 最后一层          tepD = diff          for i in range(len(grad\_list) - 2, -1, -1):              tepD = (np.dot(tepD, theta\_list[i + 1]) \* Activation\_Function\_grad(Z\_list[i + 1]))[:, 1:]              D = np.dot(tepD.T, A\_list[i])              grad\_list[i] = grad\_list[i] + D  # 增广的第一行应该去掉      for i in range(len(grad\_list)):          grad\_list[i] = grad\_list[i] / m      # 反向传播，额外计算正则化项(L2)      for i in range(len(theta\_list)):          para = theta\_list[i]          para[:, 0] = np.zeros((para[:, 0]).shape)  # 不计算第一列的偏置          grad\_list[i] = grad\_list[i] + reg\_const  \* para      vec = unroll(grad\_list)#展平      return vec |

如上代码，将所有参数视作一个矩阵考虑，梯度下降法只需遍历样本，然后分别求出每个样本对应的梯度，再求平均值即可。对于神经网络，需要知道其结构，然后使用链式法则一步一步求出。

白板上的文字

描述已自动生成

如上图是我当时写梯度下降的手稿，一步一步理清楚每一步的变换即可。期间遇到的最大问题见实验总结[4.1.3 神经网络反向传播时如何处理bias](#_神经网络反向传播时如何处理bias)

同时，我还额外实现了神经网络的正则化项（L2正则化）。

### Adam算法

|  |
| --- |
| def My\_Adam(self, X, y, alpha=0.5, beta1=0.9, beta2=0.999,epsilon=1e-8, max\_iter=3000):          m = 0          v = 0          for it in range(1, max\_iter):              g = gradient(self.theta,self.network\_struct,self.reg\_const,X, y)              m = beta1 \* m + (1 - beta1) \* g              v = beta2 \* v + (1 - beta2) \* (g\*\*2)              m2 = m / (1 - beta1\*\*it)              v2 = v / (1 - beta2\*\*it)              update\_theta=alpha \* (m2 / (np.sqrt(v2) + epsilon))              self.theta = self.theta - update\_theta |

手写实现了Adam算法，用于神经网络的反向传播算法。

# 实验过程

## 环境说明：

### 操作系统

Windows 10

### 开发语言

Python 3.8.3

### 开发环境及具体版本

见本目录下requirement.txt

### 核心使用库

numpy sklearn pandas scipy

## 源代码文件清单，主要函数清单

文件清单：

|  |  |
| --- | --- |
| 文件 | 作用 |
| requirement.txt | 说明文件，注明所用库的版本号 |
| Concrete\_Data.xls | 数据集 |
| Concrete.py | 主文件，使用各个模型进行回归拟合 |
| Data\_preprocessing.py | 数据预处理 |
| EDA.py | 探索性数据分析 |
| Regression.py | 交叉验证和绘制函数库 |
| DecisionTreeRegressorHandWrite.py | 手写决策树 |
| RandomForestHandWrite.py | 手写随机森林 |
| MLPHandWrite.py | 手写多层感知机 |
| MyMLPModel.txt | 多层感知机模型参数，存于本地便于读取 |
| LinearModelHandWrite.py | 手写传统线性回归、LASSO回归、Ridge回归 |

函数清单：

|  |  |  |
| --- | --- | --- |
| 函数 | 作用 | 所属文件 |
| main | 实例化各回归模型并进行实验，主函数 | Concrete.py |
| Regression | 采用K折训练法对传入的模型进行训练与评估。 | Regression.py |
| Draw | 绘制模型特征图像 | Regression.py |
| Data\_preprocessing | 数据预处理，正则化和归一化 | Data\_preprocessing.py |
| outlier\_test | 数据离群值甄别 | Data\_preprocessing.py |
| fit | 模型的拟合训练函数 | 所有模型 |
| predict | 模型的预测函数 | 所有模型 |
| score | 模型的评估函数 | 所有模型 |
| selectBestFeature | 选取左右子树最优的分割特征和分割值 | RandomForestHandWrite.py |
| addTree | 向随机森林模型中添加单棵决策树 | RandomForestHandWrite.py |
| createForeCast | 获取单棵树的预测结果 | RandomForestHandWrite.py |
| selectBestFeature | 选取特征边界 | RandomForestHandWrite.py |
| createtree | 搭建决策树 | RandomForestHandWrite.py |
| \_get\_split\_mse | 对传入的特征值的分割点计算mse，返回该分割点的mse便于之后选择最佳分割点 | DecisionTreeRegressorHandWrite.py |
| \_choose\_split | 遍历特征某一列的所有的不重复的点，找出MSE最小的点作为最佳分割点 | DecisionTreeRegressorHandWrite.py |
| \_choose\_feature | 遍历所有特征，计算最佳分割点对应的MSE，找出MSE最小的特征对应的分割点 | DecisionTreeRegressorHandWrite.py |
| get\_rules | 获取最终决策树每次分裂的规则 | DecisionTreeRegressorHandWrite.py |
| cost\_func | 计算损失函数 | MLPHandWrite.py |
| gradient | 利用链式法则计算梯度 | MLPHandWrite.py |
| My\_Adam | 手写的Adam算法 | MLPHandWrite.py |
| Forward\_propagation | 前向传播，计算结果 | MLPHandWrite.py |

## 实验过程

### 数据预处理

#### 探索性数据分析（EDA）

本数据集（Concrete Compressive Strength Data Set）含有9个变量，8个输入变量Cement、Blast Furnace Slag、Fly Ash、Water、Superplasticizer、Coarse Aggregate、Fine Aggregate (All above features measured in 和Age(in days)，一个输出变量Concrete Compressive Strength (Mpa)。

以下分析内容及图表绘制部分参考自文献[1]和[2]。

##### 各变量间的关系

绘制各变量两两之间的关系如下图。



为更精确地获取各变量之间相关性，我们将9个变量的相关系数绘制热力图。

手机屏幕的截图

描述已自动生成

观察可得Concrete Compressive Strength与Cement、Superplasticizer、Age有较强的相关性。Superplasticizer与Water有较强的负相关性，与Fly ash有正相关性。Fine aggregate与Water有较强的负相关性。

进一步绘制散点图来探究其中的关系。

图表, 散点图

描述已自动生成

观察上图得在其他同等条件下，Compressive strength随Cement的增加而增加，随Age的增加而增长；Age小时需要更大的Cement以获得更大的Compressive strength；Age越大需要的Water越多；Water减少时Compressive strength会增加。

图表, 散点图

描述已自动生成

观察上图得总体呈现出Fly Ash下降时Compressive strength增加、Compressive strength随着Superplasticizer增加而增加的趋势。

图表, 散点图

描述已自动生成

同前面得出的结论相同，Water减少时Compressive strength会增加，Compressive strength随着Superplasticizer增加而增加。同时在同等强度下当Water更小，Superplasticizer更多时往往需要更大的Fine aggregate。

各输入变量与输出变量的成对关系如下图所示。

图表, 散点图

描述已自动生成

可以观察到在Fly Ash=0，Superplasticizer=0，Age=0时，Concrete Compressive Strength图像呈现明显的一条直线，这部分数据可能存在异常，将此部分数据剃除。

图表, 散点图

描述已自动生成

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Coefficients | 标准误差 | t Stat | P-value | Lower 95% | Upper 95% |
| Intercept | -52.1296 | 44.1882 | -1.17972 | 0.23941 | -139.225 | 34.96569 |
| X Variable 1 | 0.118393 | 0.015707 | 7.537824 | 1.3E-12 | 0.087436 | 0.149351 |
| X Variable 2 | 0.12644 | 0.025697 | 4.920355 | 1.71E-06 | 0.07579 | 0.177089 |
| X Variable 3 | 0.040332 | 0.02621 | 1.538761 | 0.125326 | -0.01133 | 0.091992 |
| X Variable 4 | -0.08664 | 0.056356 | -1.53742 | 0.125655 | -0.19772 | 0.024435 |
| X Variable 5 | 0.083103 | 0.19955 | 0.416451 | 0.677494 | -0.31021 | 0.476417 |
| X Variable 6 | 0.01625 | 0.016618 | 0.977878 | 0.329229 | -0.0165 | 0.049004 |
| X Variable 7 | 0.042877 | 0.01924 | 2.228522 | 0.026876 | 0.004955 | 0.080799 |
| X Variable 8 | 0.384397 | 0.024232 | 15.8634 | 4.33E-38 | 0.336636 | 0.432158 |

但清洗后仅剩225组数据，且相关性的显著水平明显下降，故不可取，决定舍弃这一批次的数据。

##### 显著性水平分析

在Excel内对数据集进行回归分析，观察p值，以0.05为显著水平的阈值，以0.01为非常显著的阈值，可去除p值大于0.05的输入变量6、7，即Coarse Aggregate和Fine Aggregate。其他输入变量均与输出变量有非常显著的线性相关性。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Coefficients | 标准误差 | t Stat | P-value | Lower 95% | Upper 95% |
| Intercept | -23.16375581 | 26.5884205 | -0.871197137 | 0.383851 | -75.33795207 | 29.01044 |
| X Variable 1 | 0.119785255 | 0.008489368 | 14.11003213 | 1.96E-41 | 0.103126652 | 0.136444 |
| X Variable 2 | 0.103847249 | 0.010136216 | 10.24516947 | 1.63E-23 | 0.083957052 | 0.123737 |
| X Variable 3 | 0.087943082 | 0.012585113 | 6.987865986 | 5.03E-12 | 0.063247439 | 0.112639 |
| X Variable 4 | -0.150297904 | 0.040179254 | -3.740684259 | 0.000194 | -0.22914126 | -0.07145 |
| X Variable 5 | 0.290686943 | 0.093459877 | 3.110285971 | 0.001921 | 0.107291546 | 0.474082 |
| X Variable 6 | 0.018030184 | 0.009394211 | 1.91928665 | 0.055227 | -0.000403984 | 0.036464 |
| X Variable 7 | 0.020154456 | 0.010702688 | 1.883120981 | 0.059968 | -0.000847323 | 0.041156 |
| X Variable 8 | 0.11422562 | 0.005427493 | 21.04574138 | 5.84E-82 | 0.103575303 | 0.124876 |

##### Pearson相关系数分析

如下表，特征之间的皮尔逊相关系数矩阵表明自变量（特征）之间似乎没有高相关性。

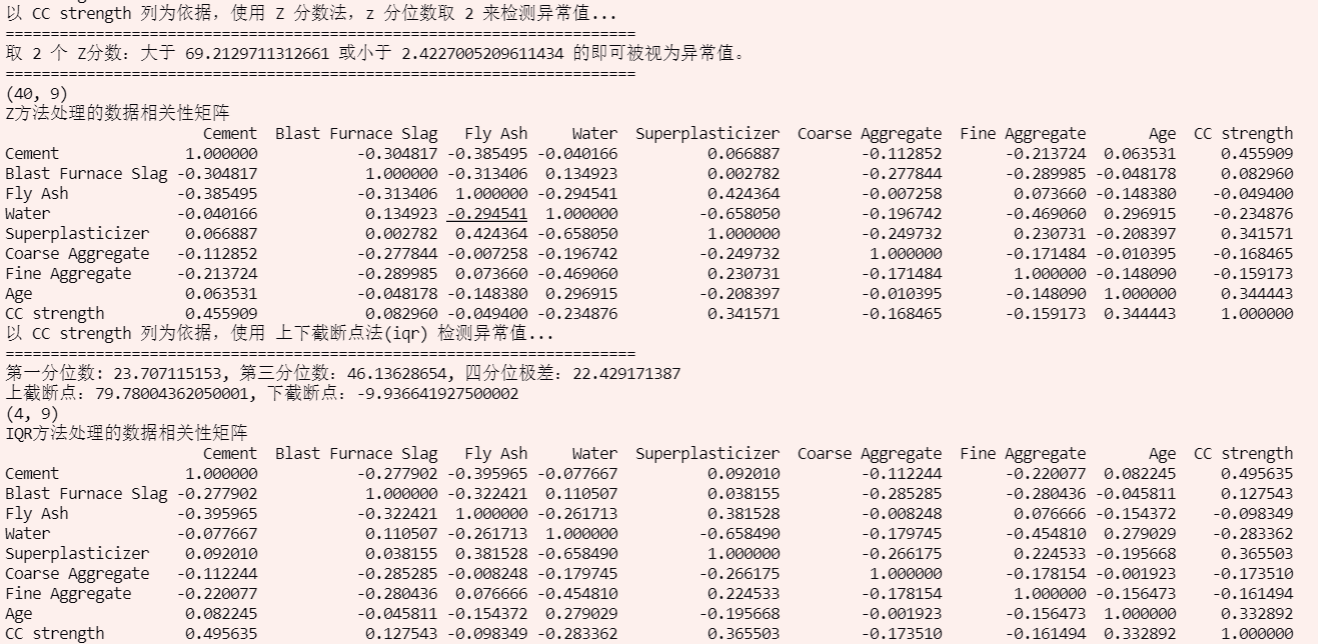
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 列 1 | 列 2 | 列 3 | 列 4 | 列 5 | 列 6 | 列 7 | 列 8 | 列 9 |
| 列 1 | 1 |  |  |  |  |  |  |  |  |
| 列 2 | -0.27519 | 1 |  |  |  |  |  |  |  |
| 列 3 | -0.39748 | -0.32357 | 1 |  |  |  |  |  |  |
| 列 4 | -0.08154 | 0.107286 | -0.25704 | 1 |  |  |  |  |  |
| 列 5 | 0.092771 | 0.043376 | 0.37734 | -0.65746 | 1 |  |  |  |  |
| 列 6 | -0.10936 | -0.284 | -0.00998 | -0.18231 | -0.2663 | 1 |  |  |  |
| 列 7 | -0.22272 | -0.28159 | 0.079076 | -0.45063 | 0.222501 | -0.17851 | 1 |  |  |
| 列 8 | 0.081947 | -0.04425 | -0.15437 | 0.277604 | -0.19272 | -0.00302 | -0.15609 | 1 |  |
| 列 9 | 0.497833 | 0.134824 | -0.10575 | -0.28961 | 0.366102 | -0.16493 | -0.16725 | 0.328877 | 1 |

#### 去除异常值（离群值）

我们选用Z-score（Z分数，）法和IQR（四分位距）法分别进行离群值甄别。

|  |
| --- |
| def outlier\_test(data, column, method=None, z=2):      if method == None:          print(f'以 {column} 列为依据，使用 上下截断点法(iqr) 检测异常值...')          print('=' \* 70)          column\_iqr = np.quantile(data[column], 0.75) - np.quantile(data[column], 0.25)          (q1, q3) = np.quantile(data[column], 0.25), np.quantile(data[column], 0.75)          upper, lower = (q3 + 1.5 \* column\_iqr), (q1 - 1.5 \* column\_iqr)          outlier = data[(data[column] <= lower) | (data[column] >= upper)]          print(f'第一分位数: {q1}, 第三分位数：{q3}, 四分位极差：{column\_iqr}')          print(f"上截断点：{upper}, 下截断点：{lower}")          return outlier, upper, lower        if method == 'z':          print(f'以 {column} 列为依据，使用 Z 分数法，z 分位数取 {z} 来检测异常值...')          print('=' \* 70)          mean, std = np.mean(data[column]), np.std(data[column])          upper, lower = (mean + z \* std), (mean - z \* std)          print(f"取 {z} 个 Z分数：大于 {upper} 或小于 {lower} 的即可被视为异常值。")          print('=' \* 70)          outlier = data[(data[column] <= lower) | (data[column] >= upper)]          return outlier, upper, lower |

甄别结果及处理后的相关性系数如下：

 由图可知，相关性系数相较处理前未有明显改善，故最终并未采用此预处理后的数据。

#### 主成分分析

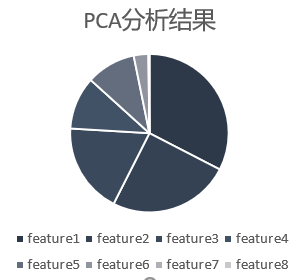
调用sklearn.decomposition的PCA模块，对上述数据集自变量进行主成分分析

|  |
| --- |
| pca=PCA(n\_components=data.shape[1])  newX = pca.fit\_transform(data)  print(pca.explained\_variance\_ratio\_) |

得到如下结果：

|  |
| --- |
| [3.25769855e-01 2.48868733e-01 1.84800583e-01 1.07660178e-01  1.00949746e-01 2.98452916e-02 1.81808433e-03 2.87530097e-04] |

作图如下：



由图可知，第7、8个特征占所有特征的方差比例为1.82e-03和2.88e-04，意味着他们几乎不含任何信息。因此，我们剔除最后两项，然后将得到的数据(1030\*6)用于后面的分析之中。剔除后所有自变量的相关关系如下：

图表, 瀑布图, 树状图

描述已自动生成

#### 归一化

我们调用sklearn preprocessing的StandardScaler模块，对自变量进行归一化，公式：

|  |
| --- |
| sc = StandardScaler()  newX = sc.fit\_transform(newX) |

由此，我们将自变量全部归一化到标准正态分布N(0,1)，然后将得到的数据(1030\*8)用于后面的分析之中。

**在下个阶段，我们将对以上方法中得到的数据分别进行分析，以进一步验证数据预处理对模型拟合产生的影响。**

### 模型拟合

#### 传统多元线性回归

首先我们直接使用最小二乘法，推导过程课件中即有，故实现即为

以data指代X,target为y的话即有

|  |
| --- |
| def fit(self,data,target):          temp = np.linalg.inv(np.matmul(data.T,data))      return np.matmul(temp ,np.matmul(data.T,target)) |

随后我们实现了基于梯度下降法的线性回归模型，同时遇到了梯度爆炸的问题。这一问题随后被我们通过自变量归一化的方法解决。具体见实验总结[4.1.1线性回归模型梯度下降法求解参数值时发生梯度爆炸](#_线性回归模型梯度下降法求解参数值时发生梯度爆炸)。

[梯度下降算法见2.4.1手写线性模型梯度下降算法。](#_手写线性模型梯度下降算法)

loss值，learning\_rate=0.01曲线：

形状

描述已自动生成

我们还曾手写mini\_batch，小批量梯度下降。但效果不尽如人意，loss曲线：

形状, 正方形

描述已自动生成放大之后：图表, 条形图

描述已自动生成

因此，此后不再使用minibatch(因为数据集比较小)

预测值与真实值对照散点图如图：

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

#### LASSO回归

此处最小二乘法的实现出了一些问题，具体见实验总结[4.1.2无法通过最小二乘法求解Lasso模型的参数值](#_无法通过最小二乘法求解Lasso模型的参数值)

因此最终仅实现了基于梯度下降法LASSO回归模型。

[梯度下降算法见2.4.1手写线性模型梯度下降算法。](#_手写线性模型梯度下降算法)

learning\_rate=0.01，=0.1，loss值曲线：

图表, 形状

描述已自动生成

预测值与真实值对照散点图如图：

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

#### Ridge回归

岭回归是在传统多元线性回归的最后加入L2正则化参数，使用最小二乘法的结论为：

以data指代X,target为y, R\_param指代的话即有

|  |
| --- |
| def fit(self, data, target, R\_param):          temp = np.matmul(data.T, data)          temp = np.linalg.inv(temp + R\_param\*np.eye(temp.shape))      return np.matmul(temp, np.matmul(data.T, target)) |

[梯度下降算法见2.4.1手写线性模型梯度下降算法。](#_手写线性模型梯度下降算法)

learning\_rate=0.01，=0.1，loss值曲线：

图表

描述已自动生成

预测值与真实值对照散点图如图：

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

#### 决策树

计算每个分割点的mse（即分割点左右两边方差之和）以选择使mse最小的特征分割点。

|  |
| --- |
| def \_get\_split\_mse(col: ndarray, label: ndarray, split: float) -> Node:          label\_left = label[col < split]          label\_right = label[col >= split]          avg\_left = label\_left.mean()          avg\_right = label\_right.mean()          mse = (((label\_left - avg\_left) \*\* 2).sum() +                 ((label\_right - avg\_right) \*\* 2).sum()) / len(label)          node = Node(split=split, mse=mse)          node.left = Node(avg\_left)          node.right = Node(avg\_right)      return node |

选择最佳分割点：

|  |
| --- |
| def \_choose\_split(self, col: ndarray, label: ndarray) -> Node:          node = Node()          unique = set(col)          if len(unique) == 1:              return node          unique.remove(min(unique))          ite = map(lambda x: self.\_get\_split\_mse(col, label, x), unique)          node = min(ite, key=lambda x: x.mse)      return node |

在各特征的最佳分割点中选择最优特征：

|  |
| --- |
| def \_choose\_feature(self, data: ndarray, label: ndarray) -> Node:          \_ite = map(lambda x: (self.\_choose\_split(data[:, x], label), x), range(data.shape[1]))          ite = filter(lambda x: x[0].split is not None, \_ite)          node, feature = min(ite, key=lambda x: x[0].mse, default=(Node(), None))          node.feature = feature      return node |

在训练模型时使用队列+广度优先搜索，控制以下条件：

1. 控制树的最大深度max\_depth；

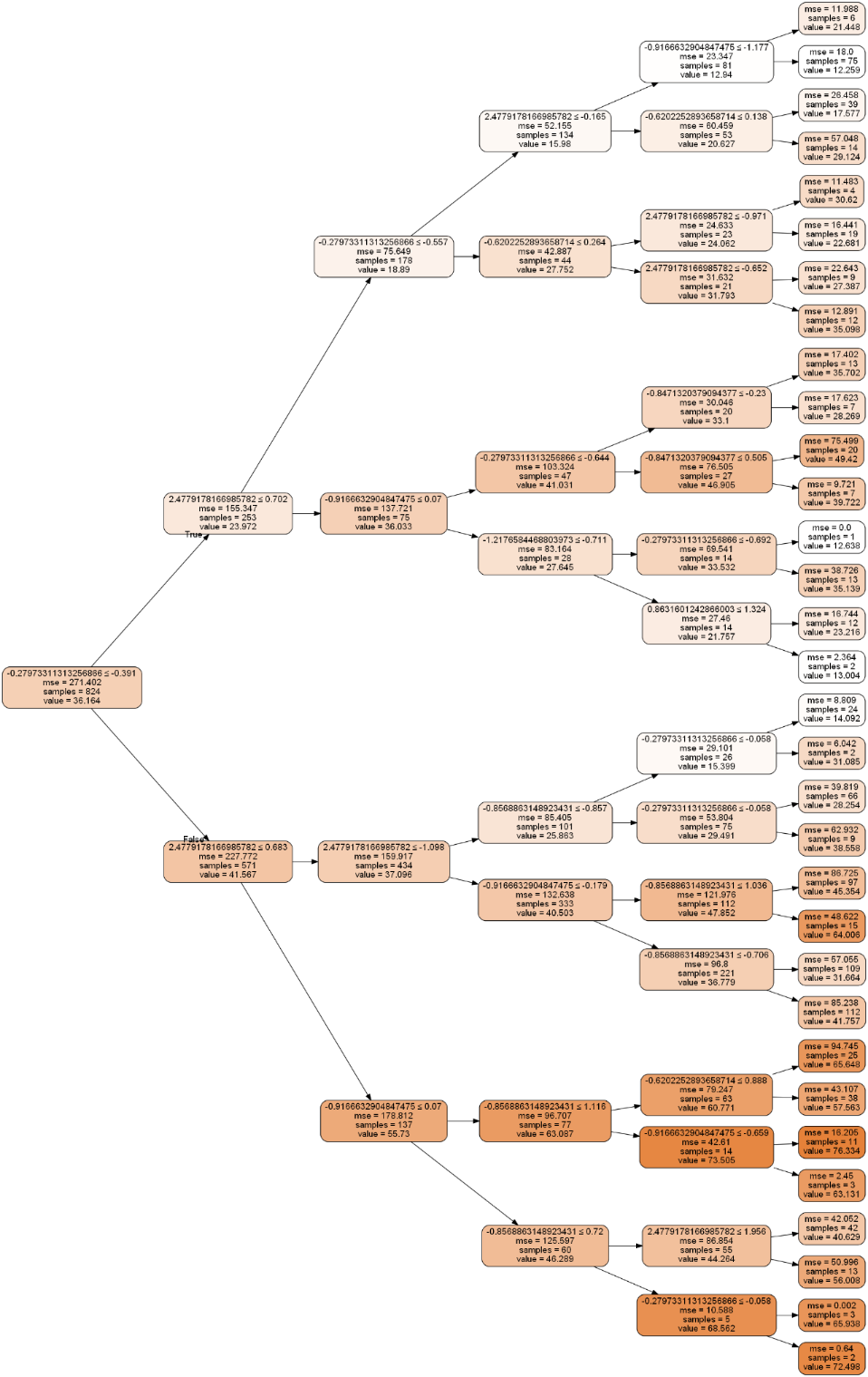
2. 控制分裂时最少的样本量min\_samples\_split；

3. 叶子结点至少有两个不重复的y值；

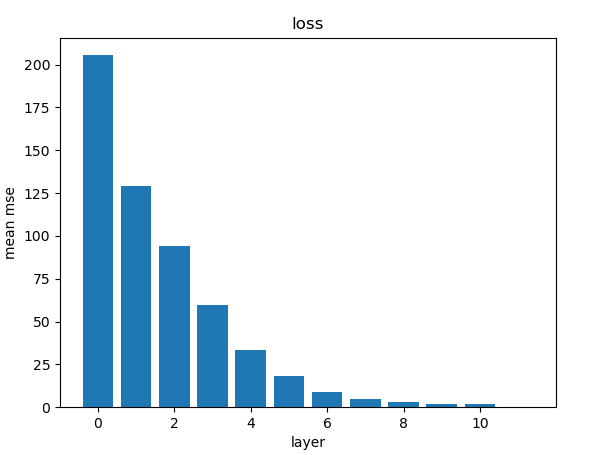
4. 至少有一个特征是没有重复值的。

|  |
| --- |
| def fit(self, data: ndarray, label: ndarray, max\_depth=12, min\_samples\_split=2):          self.\_\_init\_\_()          self.root.avg = label.mean()          que = [(self.depth + 1, self.root, data, label)]          while que:              depth, node, \_data, \_label = que.pop(0)              if depth > max\_depth:                  depth -= 1                  break              if len(\_label) < min\_samples\_split or all(\_label == label[0]):                  continue              \_node = self.\_choose\_feature(\_data, \_label) #取最优划分点对应的特征              if \_node.split is None:                  continue              node.copy(\_node)              idx\_left = (\_data[:, node.feature] < node.split)              idx\_right = (\_data[:, node.feature] >= node.split)              que.append(                  (depth + 1, node.left, \_data[idx\_left], \_label[idx\_left]))              que.append(                  (depth + 1, node.right, \_data[idx\_right], \_label[idx\_right]))          self.depth = depth      self.get\_rules() #获取当前分割规则 |

5层回归树模型可视化如下（由于篇幅限制，这里仅保留树的一半）：



通过评估score与运行时间，最终选取max\_depth为12。每层loss如下（取每层mse的均值作为loss，此处展示最后一次交叉验证的 loss结果）：



预测值与真实值对照散点图如图：

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

#### 随机森林

选择最优的特征和最佳分割点进行左右子树的划分。注意要保证内部节点和叶子节点分割需要的最小样例数，

|  |
| --- |
| def selectBestFeature(self, dataSet):      #计算特征的数目      feature\_num=dataSet.shape[1]-1      features=np.random.choice(feature\_num,self.max\_features,replace=False)      bestScore=inf; # 最好分数      bestfeature=0;     # 最优划分特征      bestValue=0;     # 最优划分特征的分割值      curScore=self.criterion.value(dataSet) # 当前未划分的目标函数得分        if shape(dataSet)[0] < self.min\_samples\_split or shape(dataSet)[0] < self.min\_samples\_leaf: # 判断样本数量是否足够划分          return None,self.getMean(dataSet)      for feature in features:          dataSet = dataSet[dataSet[:,feature].argsort()]          for index in range(shape(dataSet)[0]-1):              if index != shape(dataSet)[0]-1 and dataSet[index][feature] == dataSet[index+1][feature]:             # 排除重复值                  continue              data0 = dataSet[0:index+1, :]              data1 = dataSet[index+1:, :]              if shape(data0)[0] < self.min\_samples\_leaf or shape(data1)[0] < self.min\_samples\_leaf:                  continue;              newS=self.criterion.value(data0)+self.criterion.value(data1)              if bestScore>newS:                  bestfeature=feature                  bestValue=dataSet[index][feature]                  bestScore=newS      if (curScore-bestScore)<self.min\_change: #如果误差不大说明无法分割该子数据集          return None,self.getMean(dataSet)      return bestfeature,bestValue |

构建多棵决策树，并行化搭建随机森林模型。

|  |
| --- |
| def fit(self, X, Y):          dataSet = np.concatenate((X, Y.reshape(-1,1)), axis = -1)          self.trees=[] #确保为空再搭建森林          self.cur\_tree=0  Parallel(n\_jobs=self.n\_jobs,backend="threading")(delayed(self.addTree)(dataSet) for \_ in range(self.n\_estimators)) |

预测结果为取各决策树预测结果的均值。

|  |
| --- |
| def predict(self,X):          self.cur\_tree = 0;          l=len(X)          predict=np.mat(zeros((l,1)))          Parallel(n\_jobs=self.n\_jobs, backend="threading")(delayed(self.updatePredict)(predict, tree, X) for tree in self.trees)      #     对多棵树预测的结果取平均          predict/=self.n\_estimators          return predict |

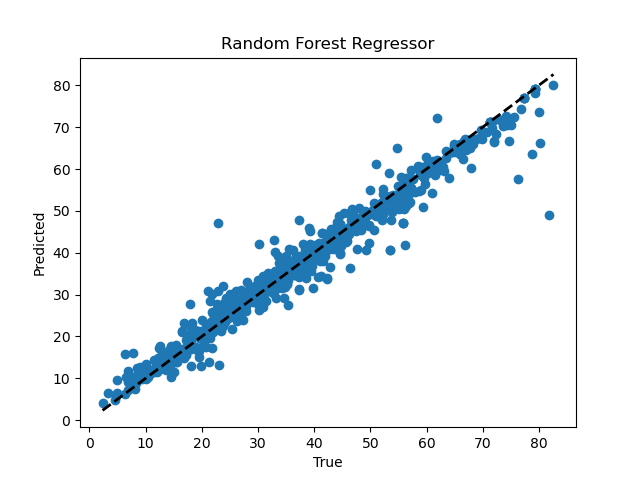
实验结果如下：

1. 调用sklearn库中的RandomForestRegresssor，选用不同标准函数的结果分别如下：
   * + criterion = “squared\_error”

五折交叉验证输出结果：

|  |
| --- |
| fold 1/5,score(R^2)=0.9226147227577468  fold 2/5,score(R^2)=0.9141392794405223  fold 3/5,score(R^2)=0.9373118467990383  fold 4/5,score(R^2)=0.9171902852970524  fold 5/5,score(R^2)=0.9055759872597926  average score(R^2): 0.9193664243108305 |

预测值与真实值对照散点图如图：

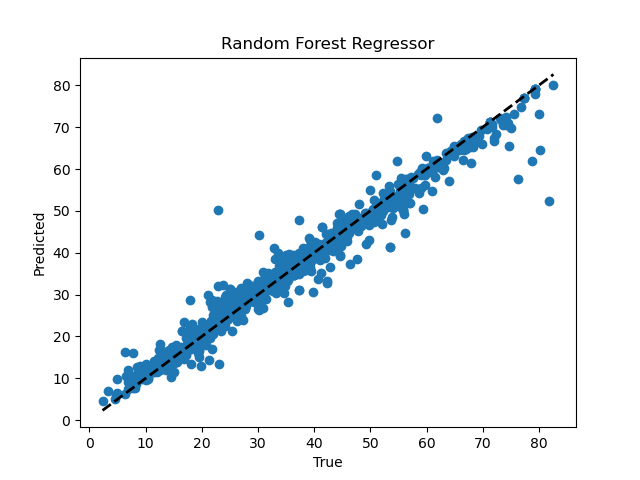


* + - criterion = “absolute\_error”

五折交叉验证输出结果：

|  |
| --- |
| fold 1/5,score(R^2)=0.9225484830359599  fold 2/5,score(R^2)=0.9119223056867468  fold 3/5,score(R^2)=0.9377286373255578  fold 4/5,score(R^2)=0.9174268571311938  fold 5/5,score(R^2)=0.9101956469513632  average score(R^2): 0.9199643860261644 |

预测值与真实值对照散点图如图：

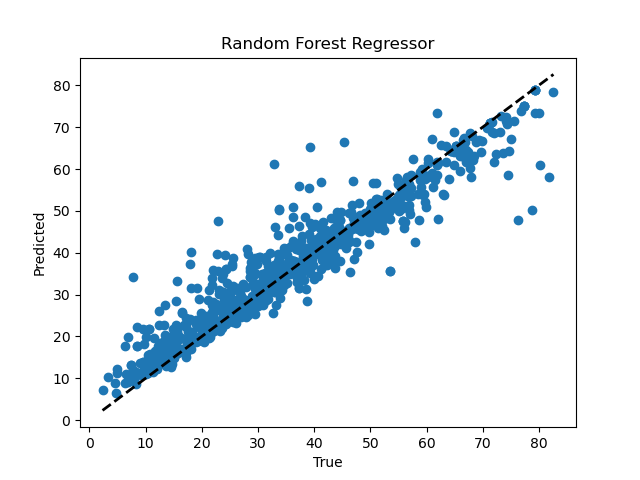


* + - criterion = “poisson”

五折交叉验证输出结果：

|  |
| --- |
| fold 1/5,score(R^2)=0.7687383051199126  fold 2/5,score(R^2)=0.7525307480413777  fold 3/5,score(R^2)=0.7546415507433242  fold 4/5,score(R^2)=0.6920888485627592  fold 5/5,score(R^2)=0.7189981629533393  average score(R^2): 0.7373995230841426 |

预测值与真实值对照散点图如图：

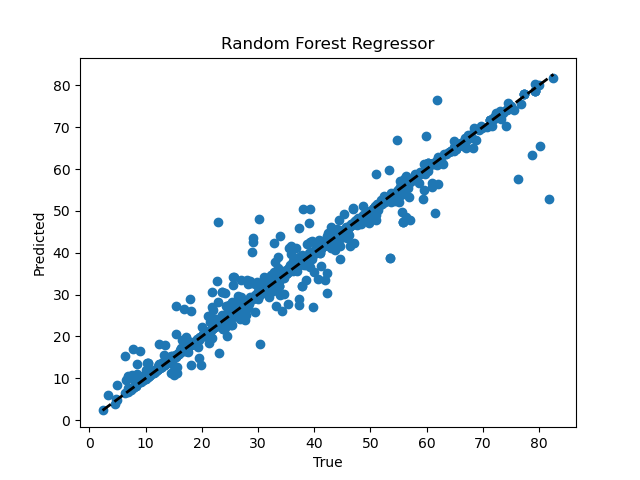


1. 手写实现结果如下：

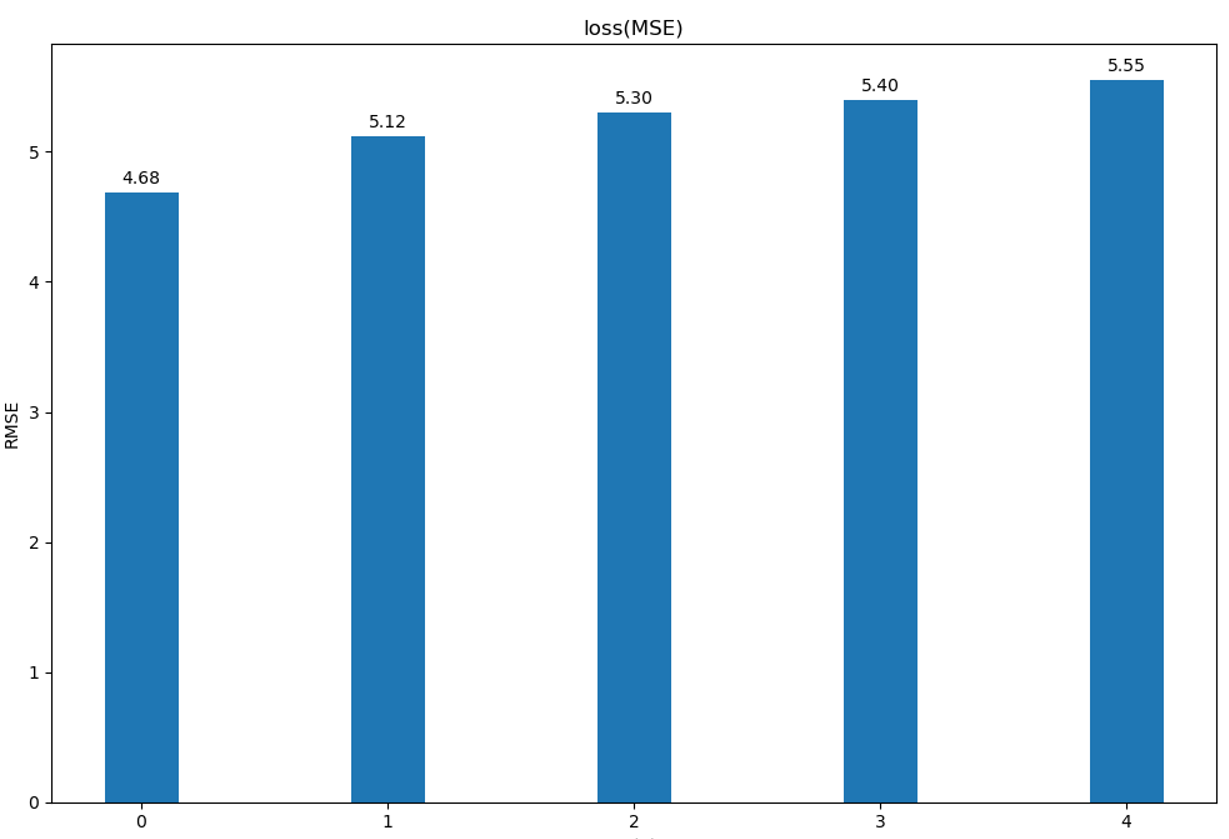
五折交叉验证输出结果：

|  |
| --- |
| fold 1/5,score(R^2)=0.8957345179201245  fold 2/5,score(R^2)=0.8975032483307323  fold 3/5,score(R^2)=0.9197340038194519  fold 4/5,score(R^2)=0.8946980681119071  fold 5/5,score(R^2)=0.9022704596029106  average score(R^2): 0.9019880595570253 |

预测值与真实值对照散点图如图：



交叉验证的loss值如下图：



#### 神经网络(多层感知机)

此处梯度下降算法的难点在于如何处理输入变量x的增广。因为我们很容易求得（它就是cost对前一层x的偏导），只要我们求得，就可以求出，之后再对求出一次偏导即可求出对应梯度。但事实上可以不求。详见实验总结[4.1.3 神经网络反向传播时如何处理bias](#_神经网络反向传播时如何处理bias)

事实上，我在这里经历了广泛的调参，开了四个文件夹。

图形用户界面, 文本, 应用程序

描述已自动生成

基本上经过调参可知，网络模型不需要过于复杂，两层足矣。也许是模型不算复杂，训练集不够大，最后选用的模型结构是9\_5(hidden), learning\_rate=0.003, 激活函数选择sigmoid(relu的训练效果远远不如sigmoid), 训练方法选用传统方法，即标准梯度下降(Adam方法效果并没有传统方法好), 一轮训练所有样本(而非如SDG和mini\_batch那样随机训练)

其中一个文件夹内的文件树如图所示：

图片包含 文本

描述已自动生成

在此简单画出五个R2较高的loss值曲线：

网络结构(hidden):7\_7\_5,

learning\_rate=0.003,

激活函数使用relu,

更新方法使用adam方法

形状

描述已自动生成

网络结构(hidden):9\_5,

learning\_rate=0.003,

激活函数使用relu,

更新方法使用标准梯度下降

图表

描述已自动生成

网络结构(hidden):9\_5,

learning\_rate=0.003,

激活函数使用relu,

更新方法使用adam方法

图表

描述已自动生成

网络结构(hidden):9\_5,

learning\_rate=0.003,

激活函数使用sigmoid,

更新方法使用传统方法

图表, 形状

描述已自动生成

网络结构(hidden):9\_5,

learning\_rate=0.003,

激活函数使用sigmoid,

更新方法使用adam方法

图表

描述已自动生成

预测值与真实值对照散点图如图（使用效果最好的一次）：

图表, 散点图

描述已自动生成 图表, 散点图

描述已自动生成

上述每一次的训练都耗时及其久（>1e6ms）

提交代码时，我将手写的神经网络模型的fit函数从训练改为了加载。

它会加载我存储在model文件夹内的模型，以减少训练时间。

如需训练，可向fit方法中传入isload=False，即可正常训练。

### 模型比较

#### MSE（均方差）

图片包含 表格

描述已自动生成

#### R2\_score

图片包含 图表

描述已自动生成

采用老师默认的验证函数Regression，参数为5折验证，测试集占20%

结果如下：

线性模型 决策树/随机森林

报纸上的文字

中度可信度描述已自动生成 报纸上的文字

中度可信度描述已自动生成

神经网络

报纸上的文字

中度可信度描述已自动生成

#### 训练时间

手写神经网络所需时间>1e6，其他模型训练时间如下：

表格

中度可信度描述已自动生成

## 实验结论

由此可知，决策树和随机森林模型在该数据集上的拟合效果最好，经过合理调参且对数据进行过归一化后的多层感知机次之，三种线性模型最差。（但神经网络训练速度最慢，随机森林次之，决策树再次之，线性模型最快）

主成分分析对三种线性模型及多层感知机的拟合并无十分显著的影响（这可能是因为数据集规模较小引起的），而决策树和随机森林模型在使用经PCA降维后的数据后拟合效果反而变差了。

而对数据进行归一化后，多层感知机和基于梯度下降方法实现的三种线性模型的拟合效果产生了巨大提升，其他模型（包括基于最小二乘法实现的两种线性模型）的拟合效果则没有什么显著变化。

# 总结

## 实验中存在的问题及解决方案

### 线性回归模型梯度下降法求解参数值时发生梯度爆炸

在手写线性模型梯度下降法时，我们发现，每一次迭代后参数值均发生指数级的增长，迭代个几百轮后所有参数均因突破Inf变成Nan了。

梯度爆炸(参数) 梯度爆炸(loss值)

报纸上的文字

描述已自动生成 文本

描述已自动生成

梯度爆炸(loss值)

注意以下图像中y轴是loss的对数

图表, 折线图

描述已自动生成

**解决方案**：我们首先怀疑是学习率过高，将学习率从0.001调至1e-7，终于避免了梯度爆炸的问题。但此时由于学习率过低，而且有时也有概率发生梯度爆炸，远不如最小二乘法。

这样显然是不行的。我们思考发现是因为自变量的值过大，每次参数值与自变量相乘时均得到一个较大的梯度。我们尝试将所有自变量**归一化至标准正态分布**后再进行训练，成功解决这一问题！学习率也调整至0.01，只需300-400轮即可收敛，达到跟最小二乘法一样的效果。

### 无法通过最小二乘法求解Lasso模型的参数值

在进行线性模型Lasso的最小二乘法求解参数值时，我们注意到以下方程无法求解

其中X是一个m×n的矩阵，y是一个m×1的矩阵，是常数，是我们的待求变量，一个n×1的矩阵。

我们注意到z(x)即绝对值函数在x=0处不可微，且我们对分段函数如何分离g()束手无策，故无法通过最小二乘法求解Lasso模型的参数。

**解决方案**：我们决定采用梯度下降法求解Lasso模型的参数值，并最终成功。

### 神经网络反向传播时如何处理bias

如图是hidden为2层的神经网络示意图。在我们手写神经网络的过程中，为求方便我们将所有待运算的自变量x（一个1×n的向量）进行了一个增广处理（即在前面拼接一列1,使x’=(1,x)），以便在前向传播时可以直接利用矩阵乘法进行运算。但在反向传播时如何处理x’与x的关系便成了一个难题。

图示

描述已自动生成

**解决方案**：

我们容易求得（它就是cost对前一层x的偏导，最后一层即为predict-y）

容易知道，只要我们求得，就可以根据的值及链式法则求出，之后再对求出一次偏导即可求出对应梯度。

那么最朴素想法就是求得。可以令x右乘一个n×(n+1)的矩阵，即x’=xM，从而得到x和x’的线性关系，再利用这一关系得到最后利用链式法则处理x。但是这一方法过于复杂，会大大增加计算量。

注意到前向传播时增广的值始终为1（因为bias），是一个常数。故求取时可以不使用链式法则先求，再求，而是将梯度矩阵的第一列直接去掉，即可得到。（不要用矩阵求偏导的角度考虑，而是从矩阵中一个元素的角度考虑，就能想到这点。因为梯度矩阵除了第一列外，所有元素均是的运算结果，只有第一列是因为增广得到的额外列）

由此，绕开了求，简单方便地解决了 的求取问题。

## 后续改进方向

### 提升模型训练速度

1、可利用numpy.gpu提升模型训练速度(见<https://gitee.com/killf/numpy.gpu>)；

或直接使用支持gpu加速的pytorch库代替numpy。

2、尝试使用C/C++ 等更接近底层的编程语言重写梯度下降算法，提升计算速度。

### 尝试使用更多模型拟合

我们本次实验只采用了七个模型拟合这些数据，并成功基于numpy自己手写实现了这些模型。此后可以尝试使用SVM、KNN、KRR等模型进行拟合，并尝试手写这些模型，以观察拟合效果。

# 参考文献

[1] <https://github.com/pranaymodukuru/Concrete-compressive-strength>

[2] <https://github.com/nitya123-github/Concrete-strength>

[3] <https://www.bilibili.com/video/BV16x411V7Qg>