# 机器学习大作业一：回归分析



# 1 实验概述

## 1.1 实验目的

回归指研究一组随机变量$(Y_1, Y_2, \ldots, Y_i)$与另一组随机变量$(X_1, X_2, \ldots, X_k)$之间关系的统计分析方法。当因变量和自变量为线性关系时，它是一种特殊的线性模型。回归分析是通过规定因变量和自变量来确定变量之间的因果关系，建立回归模型，并根据实测数据来求解模型的各个参数，然后评价回归模型是否能够很好地拟合实测数据。

本实验通过在数据集上训练、测试线性回归模型学习线性回归模型的具体使用过程。通过调整学习率、正则化参数、数据维度变换等查看其对回归效果的影响。根据线性回归模型原理以及梯度下降原理编写线性回归算法，掌握线性回归内部原理。

## 1.2 实验内容

**①通过scikit-learn库进行线性回归实验**

通过修正后的Boston房价数据集进行线性回归。包括数据清洗、数据预处理、回归模型建立、模型训练与测试、查看预测效果等。

**②学习率与正则化参数分析**

在上述实验的基础上，调整模型的学习率以及正则化参数，查看二者对模型训练效果的影响。

**③数据集进一步分析**

对上述实验中的数据集进行进一步分析与实验。包括查看输入变量特征之间的相关性，进行数据升维与降维并重新进行训练，查看调整后的训练效果。

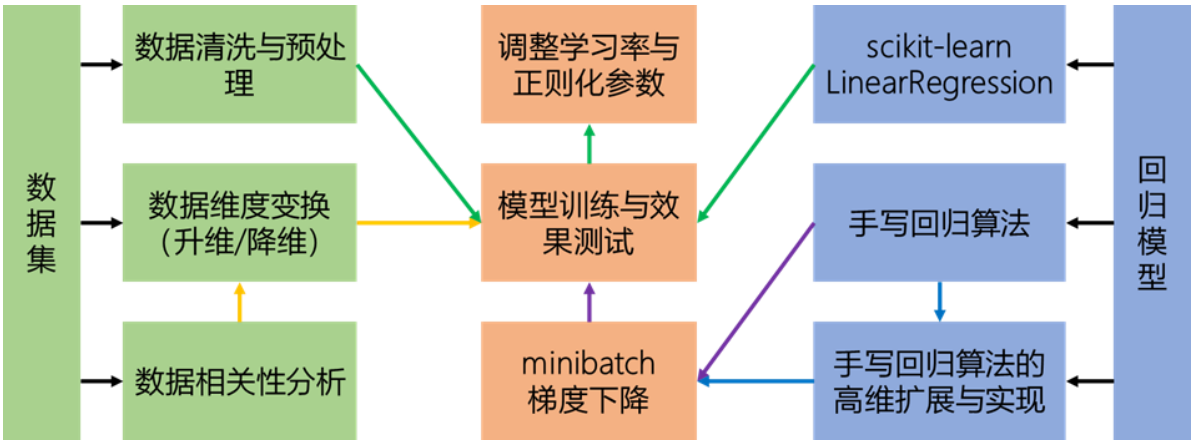
**④手写实现线性回归算法**

根据线性回归模型原理手写线性回归算法。包括参数的求导与更新、mini batch随机梯度下降算法的实现，并加入可选择的学习率与正则项，同时将模型扩展到更高维度的线性回归算法，并进行实现。将模型封装为线性回归类。

# 2 实验方案设计

## 2.1 总体设计思路与总体架构

实验总体设计架构如下图所示。




其中不同矩形颜色代表实验的不同模块，分别为数据集模块、模型模块、训练与优化模块。不同颜色的箭头代表实验中的不同步骤。

## 2.2 核心算法及基本原理

### 1 线性回归

线性回归算法即通过线性模型

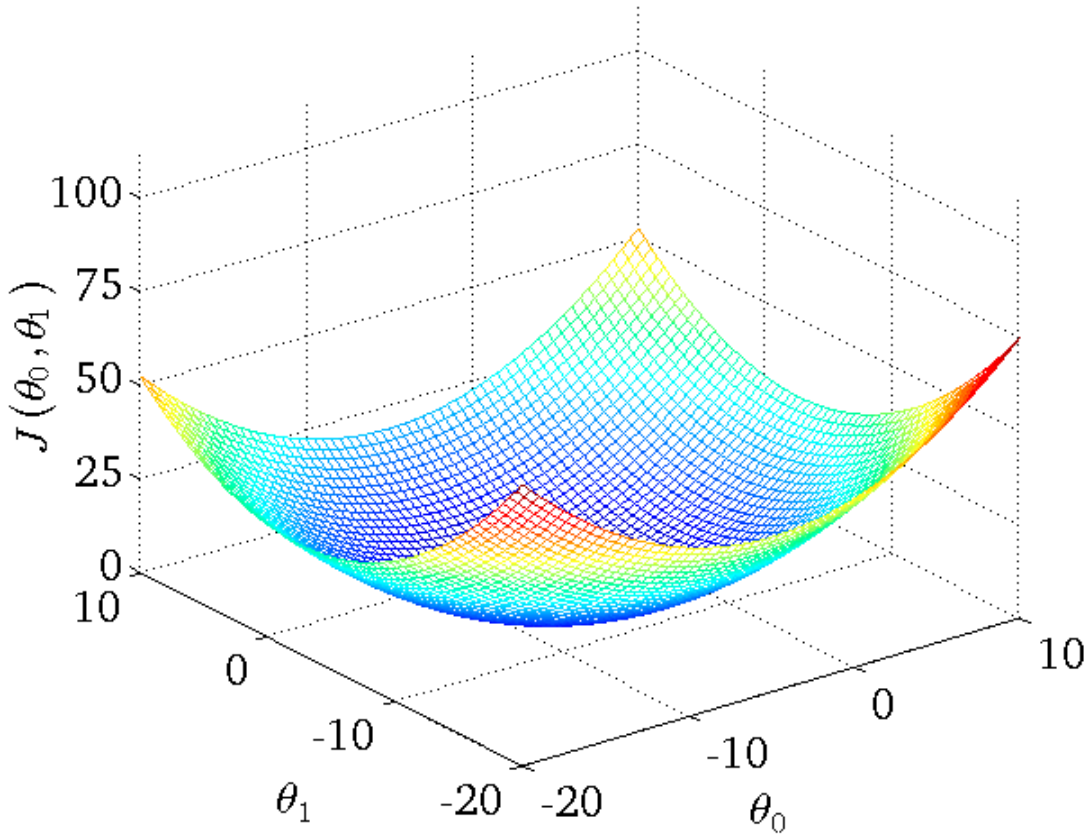
$$y = f_\theta(x) = \theta_0 + \sum_{j=1}^{d} \theta_j x_j = \theta^T x,$$

$$where\ x = (1, x_1, \ldots, x_d).$$

来拟合随机变量$x_i$与因变量$y$之间的关系。拟合的目标是使得预测值与观测值之间的误差最小，即求$\theta$使得

$$min_\theta \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_\theta(x_i)).$$

其中的误差函数$L$可取$L(y_i, f_\theta(x_i)) = (y_i - f_\theta(x_i))^2$。求解该函数最小值的方法可使用最小二乘法直接进行计算，但由于计算过程中需要计算矩阵的逆，而当输入向量的维数较高时，计算矩阵的逆的时间开销往往较大。因此在实际应用中，通常采用梯度下降或mini batch梯度下降方法来解决。




这是因为损失函数本身是凸函数，在进行梯度下降时不会由于陷入极小值点而导致模型无法训练到最优参数。

另外，如果损失函数中只设定让损失达到最小，模型可能会出现过拟合的情况；同时由于参数的搜索空间较大，模型可能难以达到收敛。此时可以在模型中加入正则化项。

$$J(\theta) = \frac{1}{2N}[\sum_{i=1}^{N}(f_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2].$$

正则化项可以控制参数幅度，避免出现太大的搜索空间，并使得模型参数尽量简约。

## 2 决定系数

决定系数反应因变量的全部变异能通过回归关系被自变量解释的比例，常用于评价回归模型的效果。决定系数越大，拟合优度越大，自变量对因变量的解释程度越高，自变量引起的变动占总变动的百分比高，观察点在回归直线附近越密集。在总体回归分析中，决定系数的定义公式如下。

$$R^2 = \frac{\sum(\hat{y} - \bar{y})^2}{\sum(y - \bar{y})^2} = 1 - \frac{\sum(y - \hat{y})^2}{\sum(y - \bar{y})^2}.$$

## 2.3 模块设计

### 1 通过scikit-learn模块进行线性回归实验（刘旭东）

#### ①数据清洗与数据预处理

本次实验选择**修正后的波士顿房价数据集**。修正后的数据集相比于原来的波士顿房价数据集来说修正了一些小错误，并增加了观测数据的经度与纬度值。数据集下载链接：修正波士顿房价数据集。在修正后的数据集上进行训练，还可以与原数据训练效果进行对比。在代码中保留了针对两种数据集的训练与测试过程，可通过调整参数进行选择。

这里只展示修正后的数据集的预处理与数据清洗。首先读入数据，数据存储在xls文件中，通过pandas的read_excel函数进行读取。

```
if CORRECTED_BOSTON: #选择修正数据集
    path='data/boston_corrected.xls'
    sheet=pd.read_excel(io=path)
    X=sheet.copy()
    X.drop(columns=['MEDV','OBS','TOWN','CMEDV'],inplace=True) #去掉标签数据、序号等特征
    y=sheet['MEDV'] #取出标签
```

通过dataframe的describe函数查看数据情况。

| | OBS | TOWN.1 | TRACT | LON | LAT | MEDV | CMEDV | CRIM | ZN | INDUS | CHAS | NOX |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 |
| mean | 253.500000 | 47.531621 | 2700.355731 | -71.056389 | 42.216440 | 22.532806 | 22.528854 | 3.613524 | 11.363636 | 11.136779 | 0.069170 | 0.554695 |
| std | 146.213884 | 27.571401 | 1380.036830 | 0.075405 | 0.061777 | 9.197104 | 9.182176 | 8.601545 | 23.322453 | 6.860353 | 0.253994 | 0.115878 |
| min | 1.000000 | 0.000000 | 1.000000 | -71.289500 | 42.030000 | 5.000000 | 5.000000 | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 |
| 25% | 127.250000 | 26.250000 | 1303.250000 | -71.093225 | 42.180775 | 17.025000 | 17.025000 | 0.082045 | 0.000000 | 5.190000 | 0.000000 | 0.449000 |
| 50% | 253.500000 | 42.000000 | 3393.500000 | -71.052900 | 42.218100 | 21.200000 | 21.200000 | 0.256510 | 0.000000 | 9.690000 | 0.000000 | 0.538000 |
| 75% | 379.750000 | 78.000000 | 3739.750000 | -71.019625 | 42.252250 | 25.000000 | 25.000000 | 3.677083 | 12.500000 | 18.100000 | 0.000000 | 0.624000 |
| max | 506.000000 | 91.000000 | 5082.000000 | -70.810000 | 42.381000 | 50.000000 | 50.000000 | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 |

可以看到数据中共有506条，不同数据的数量级有所差异，需要在后面进行归一化处理，避免在训练过程中受数量级差异的影响。

通过isna()函数查看数据中是否有缺失值。

```
sheet.isna().any() #输出全为False，说明没有缺失值
```

查看各属性的dtype，初步判断是否有类型异常属性值。得到的结果如下，除了TOWN属性（字符串类型）以外其余属性都为统一的数值型数据。

```
属性        类型        属性意义
TOWN      object       name of town
TOWN.1    int64        number of town
TRACT     int64        enumerate value of tract
LON       float64      longitude of observation
LAT       float64      latitude of observation
MEDV      float64      Median value of owner-occupied homes in $1000's
CMEDV     float64      Median value of owner-occupied homes in $1000's
CRIM      float64      per capita crime rate by town
ZN        float64      proportion of residential land zoned for lots over 25,000
sq.ft.
INDUS     float64      proportion of non-retail business acres per town
```

```
CHAS    int64       Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
NOX     float64     nitric oxides concentration (parts per 10 million)
RM      float64     average number of rooms per dwelling
AGE     float64     proportion of owner-occupied units built prior to 1940
DIS     float64     weighted distances to five Boston employment centres
RAD     int64       index of accessibility to radial highways
TAX     int64       full-value property-tax rate per $10,000
PTRATIO float64     pupil-teacher ratio by town
B       float64     1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
town
LSTAT   float64     % lower status of the population
```

下面进行数据归一化。首先将数据集划分成训练集和测试集。

```
#划分训练集测试集
if CORRECTED_BOSTON:

 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=2
7)
```

通过sklearn的归一化方法进行归一化。

```
#特征归一化
from sklearn.preprocessing import MinMaxScaler
#转换为二维数组方便拟合
if CORRECTED_BOSTON:
    y_train=y_train.values.reshape(-1,1)
    y_test=y_test.values.reshape(-1,1)
scalerX=MinMaxScaler().fit(X_train) #训练数据
scalery=MinMaxScaler().fit(y_train) #标签
X_train,X_test=scalerX.transform(X_train),scalerX.transform(X_test)#进行归一化
y_train,y_test=scalery.transform(y_train),scalery.transform(y_test)
y_train=y_train.reshape(-1,)#调整为训练需要的形状
y_test=y_test.reshape(-1,)
```

**②建立回归模型并进行训练**

通过sklearn中的LinearRegression进行线性回归训练。在训练函数中调用cross_val_score进行k折交叉验证，计算平均得分。

```
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score,KFold,ShuffleSplit
from scipy.stats import sem
from sklearn import linear_model

def train_score(regr,X_train,y_train):
    regr.fit(X_train,y_train)
    print('训练集:')
    print('决定系数: ',regr.score(X_train,y_train))

    cv=KFold(5,shuffle=True,random_state=27)#定义k折交叉验证
    scores=cross_val_score(regr,X_train,y_train,cv=cv)#进行交叉验证计算
    print('K折交叉验证 平均决定系数: {0}(+/-
{1})'.format(np.mean(scores),sem(scores)))#输出平均值与标准差
regr=linear_model.LinearRegression()
```

```
train_score(regr,X_train,y_train)
```

或者也可以自定义k折交叉验证函数，在自定义函数中可以加入每一折的预测图像并计算决定系数。

```
#自定义k折交叉验证
def regression(model,data,target,splits=5,size=0.2):
        shuffle = ShuffleSplit(n_splits=splits, test_size=size, random_state=7)#
随机打乱数据顺序
        n_fold = 1
        score_all = 0
        for train_indices, test_indices in shuffle.split(data):
            print('fold {}/{}'.format(n_fold,splits))
            #获取每一折的训练与测试数据
            X_train = data[train_indices]
            y_train = target[train_indices]
            X_test = data[test_indices]
            y_test = target[test_indices]
            #模型训练并计算决定系数
            model.fit(X_train,y_train)
            score = model.score(X_test, y_test)
            #预测后打印预测结果
            result = model.predict(X_test)
            plt.plot(np.arange(len(result)), y_test,label='true value')
            plt.plot(np.arange(len(result)),result,label='predict value')
            plt.show()
            print("第{}折 决定系数 {:.3f}".format(n_fold,score))
            score_all += score
            n_fold += 1
        print("平均决定系数 ",score_all/splits)
```

**③模型测试与效果评价**

模型训练结束后，通过测试集进行回归效果测试。首先定义两个画图函数，可以将回归效果进行可视化。

```
#将预测值与观测值同时打印到图中，查看对于同一组数据的预测值与观测值的差异
def plot_predict_label(y,y_pred):
    plt.figure()
    plt.plot(y)
    plt.plot(y_pred)
    plt.legend(['MEDV','predict MEDV'])
    plt.show()

#将预测值与观测值作为横纵坐标打印到图中，并与斜率为1的直线进行比较，查看预测效果
def
visualization_for_predict_data(data_observed,data_predicted,label_observed,label
_predicted):
    plt.scatter(data_observed,data_predicted)

    max_v=np.max([data_predicted]+[data_observed])*1.1
    min_v=np.min([data_predicted]+[data_observed])*1.1

    plt.plot([min_v,max_v],[min_v,max_v],'red')

    plt.xlabel(label_observed)
```

```
    plt.ylabel(label_predicted)
    plt.show()
```

同时，计算测试集预测值与观测值的决定系数，用来衡量回归模型效果。

```
def test_measure_visualization(X,y,regr):
    y_pred=regr.predict(X)#训练后的模型进行预测
    print('测试集 决定系数：',r2_score(y,y_pred),'\n'*2)
    #可视化预测结果
    visualization_for_predict_data(y,y_pred,'y_test','y_pred')
    plot_predict_label(y,y_pred)
```

## 2 学习率与正则化参数等分析（王攀栋）

**①实验1：对比四种不同的模型训练方式**

分别使用正规方程法NE、梯度下降法BGD、正则化正规方程法reg-NE、正则化梯度下降法reg-BGD训练线性模型，利用K折交叉验证法进行评价。取K=5，输出每一折在测试集上的MSE损失值与R2评价值，并取平均。

具体代码详见myLinearRegression.py和myLR_test.py。

**②实验2：探究特征缩放与学习率设置的作用**

分别使用原始数据集、MinMax缩放后的数据、Standard标准化后的数据进行训练，观察模型训练效果。

**对于正规方程法，不同的数据处理方式不影响矩阵计算结果**，因此不需要特征缩放。

对于梯度下降法，则需要先考虑初始学习率的设置。具体实验效果详见实验结果展示。

**③实验3：绘制模型的学习曲线，探究模型拟合效果**

划分得到训练集与测试集（7:3）后，调整训练样本数，重复训练模型（分别使用正规方程和梯度下降），记录每个模型对其训练样本和测试集的预测效果。

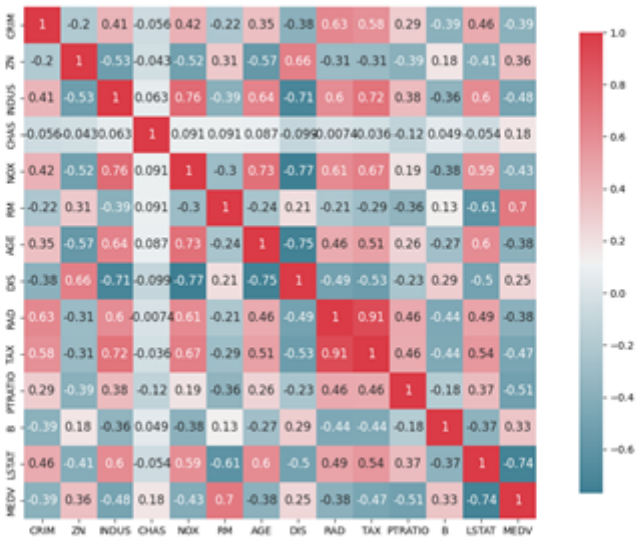
**④实验4：探讨正则项系数对模型训练的影响**

在手写模型中，只需要在正规方程和梯度下降的过程中加入正则项$\lambda \sum \theta^2$的影响即可（若不进行正则化，只需令正则项系数$\lambda = 0$）。分别令$\lambda$取[0, 0.001, 0.01, 0.1, 1]以及大于1的各个整数，绘制得到正规方程和梯度下降在不同λ下训练集与测试集的损失值变化曲线。具体实验效果详见实验结果展示。

## 3 数据集的数据变换（相关性、升维降维）与实验（王国润）

**①数据相关性的热力图分析**

为了讨论波士顿房价数据各个特征的联系和升降维的关系，我们采用了sklearn里面原始自带的数据进行控制变量的实验（即使存在上文提到的数据不准确）。当通过load boston获取对象后，通过获取data和target获取特征和标签。我们首先要对包含标签的特征进行特征分析。根据热力图进行分析，热力图如下：




我们可以从图中清楚的发现，对于标签MEDV，LSTAT和RM对于标签来说的相关度极高，经过分析后确实如此：对于低收入阶层的比例代表着总体经济水平。而每栋楼的房间数也确实满足了整体的关系：别墅（单楼单间）也确实比普通公寓楼（单楼多间）的房价贵。

此外我们也会发现某些特征之间的相关性极高，如TAX和RAD，INDUS和NOX，在后续的p-value降维中也提供了原理支持，即删除相关特征在机器学习可降低维度灾难。

这里我们使用皮尔逊相关系数系数衡量相关性：对于任意两个变量，我们计算他们的协方差和标准差的商，来衡量相关性。计算方法如下。

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}.$$

我们利用dataframe的corr函数计算，method为皮尔逊。利用seaborn和matplotlib绘图。

```
c=Boston.feature_names
c=np.append(c,'MEDV')
df=pd.DataFrame(np.concatenate([x,y[:,np.newaxis]],axis=1),columns=c)
_, ax = plt.subplots(figsize=(12, 10))  # 分辨率1200×1000
corr = df.corr(method='pearson')  # 使用皮尔逊系数计算列与列的相关性
# corr = df.corr(method='kendall')
# corr = df.corr(method='spearman')
cmap = sns.diverging_palette(220, 10, as_cmap=True)  # 在两种HUSL颜色之间制作不同的调色板。图的正负色彩范围为220、10，结果为真则返回matplotlib的colormap对象
_ = sns.heatmap(
    corr,  # 使用Pandas DataFrame数据，索引/列信息用于标记列和行
    cmap=cmap,  # 数据值到颜色空间的映射
    square=True,  # 每个单元格都是正方形
    cbar_kws={'shrink': .9},  # `fig.colorbar`的关键字参数
    ax=ax,  # 绘制图的轴
    annot=True,  # 在单元格中标注数据值
    annot_kws={'fontsize': 12})  # 热图，将矩形数据绘制为颜色编码矩阵
plt.show()
```

**②数据降维**

我们采用sklearn里面的库函数StandardScaler()进行特征缩放，即归一化。这样不仅可以使得特征间的差异变小，更重要的是在上文提到的相关性分析中，在数据标准化$(\mu = 0, \sigma = 1)$后，Pearson相关性系数、Cosine相似度、欧式距离的平方可认为是等价的。然后由于我们的特征共13个有506个样本，因此我们通过利用numpy添加常数项系数后，会变为（506，14）。直接调用statsmodel中的ols会得到一个回归器，可以查看系数以及置信度。ols指的是普通最小二乘法，即课上推导的线性最小二乘法，使散点图上的所有观测值到回归直线距离的平方和最小。对于每一个模型会生成下面的报告。

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.770
Model:                            OLS   Adj. R-squared:                  0.762
Method:                 Least Squares   F-statistic:                     93.87
Date:                Sun, 13 Mar 2022   Prob (F-statistic):          9.85e-108
Time:                        20:58:43   Log-Likelihood:                -1102.0
No. Observations:                 379   AIC:                             2232.
Df Residuals:                     365   BIC:                             2287.
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         22.4088      0.233     96.005      0.000      21.950      22.868
x1            -1.0117      0.322     -3.143      0.002      -1.645      -0.379
```

接着我们利用p-value进行特征的降维。p值是指在一个概率模型中，统计摘要（如两组样本均值差）与实际观测数据相同，或甚至更大这一事件发生的概率。换言之，是检验假设零假设成立或表现更严重的可能性。p值若与选定显著性水平（0.05或0.01）相比更小，则零假设会被否定而不可接受。然而这并不直接表明原假设正确。p值是一个服从正态分布的随机变量，在实际使用中因样本等各种因素存在不确定性。产生的结果可能会带来争议。即在原假设为真的前提下出现观察样本以及更极端情况的概率。p值可通过计算chi-square后查询卡方分布表得出，用于判断H0假设是否成立的依据。大部分时候，我们假设错误拒绝H0的概率为0.05，所以如果p值小于0.05，说明错误拒绝H0的概率很低，则我们有理由相信H0本身就是错误的，而非检验错误导致。

大部分时候p-value用于检验独立变量与输入变量的关系，H0假设通常为假设两者没有关系，所以若p值小于0.05，则可以推翻H0（两者没有关系），推出H1（两者有关系）。更为通俗的讲，一个变量的p-value小于0.05，所以这个变量很重要；反之就显得不那么重要，因此可以进行降维。代码如下。

```
x_option = x_train[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]]
ols = sm.OLS(endog= y_train,exog=x_option,).fit()
```

**③数据升维**

对于升维来说，我们先通过降维筛除了部分特征，然后再通过特征的组合，形成新的特征，再通过特征进行回归预测。通过PolynomialFeatures(degree=2)，融合形成2次特征，然后再次通过归一化，并自动增加常数项的系数，然后进行回归。

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
x = x[:,[0,1,2,4,5,6,8,9,10,11,12]]  #这里采用了上文剔除特征的结果
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =
0.25,random_state = 0)#分割数据集

pf = PolynomialFeatures(degree=2)
x_train = pf.fit_transform(x_train)#这个操作会自动增加常数项的系数，用法可以参考文档
print(pf.n_input_features_)
print(pf.n_output_features_)
```

**④二乘方法比较**

此外，我们还探寻了普通最小二乘和广义最小二乘在效果上差异。我们在同等特征类型、维度下进行模型拟合，最终r2_score变为了由上文提到的0.6369变为0.6375。因此，对于特征拟合的时候，把模型带来的误差想象成内部方差，其他因素导致的误差为外部协方差。用同一个模型拟合时，因为这模型的误差是固定的，所以每次量的方差都是一样的，所以协方差阵是对角阵，且对角线都是一样的（同一个模型的方差）。这就是普通最小二乘。当用不同的模型来拟合时，除了不同模型误差不固定，其他因素也会对这个误差造成影响，所以外部协方差不是对角阵。这就是广义最小二乘。因此不同的二乘方法会对数据进行不同的效果。

```
x_option = x_train[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]]
ols = sm.OLS(endog= y_train,exog=x_option,).fit()
print(ols.summary())   #查看回归器的参数信息
y_pre = ols.predict(x_test[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]])
print(r2_score(y_test,y_pre))

x_option = x_train[:,[0,1,2,4,5,6,8,9,10,11,12,13]]
glsar = sm.GLSAR(endog= y_train,exog=x_option,).fit()
print(glsar.summary())   #查看回归器的参数信息
y_pre = glsar.predict(x_test[:,[0,1,2,4,5,6,8,9,10,11,12,13]])
print(r2_score(y_test,y_pre))
```

## 4 线性回归算法实现（王攀栋、刘旭东）

### ①二维线性回归算法实现（王攀栋）

手写线性规划模型（含梯度下降法与正规方程法，并可选择是否引入正则项、是否动态调整学习率），并封装为MyLinearRegression类。在其中加入类似于sklearn的接口，并加入多种训练方式，包括未归一化的梯度下降训练，归一化的梯度下降训练，未归一化的正规方程法训练，归一化的正规方程法训练，将所有训练方法整合到一个fit函数中，并通过参数对训练方法进行调整。

```
def fit(self, train_x, train_y, method=2):
    """快速进入不同的训练方法"""
    if method in [1, 'ne']:
        self.fit_ne(train_x, train_y)
    elif method in [2, 'bgd']:
        self.fit_bgd(train_x, train_y)
    elif method in [3, 'ne_reg']:
        self.fit_ne(train_x, train_y, regularized=True)
    elif method in [4, 'bgd_reg']:
        self.fit_bgd(train_x, train_y, regularized=True)
```

优化器内部的算法如下。其中求导通过numpy 数组的计算实现。

```
# 梯度下降
for i in range(iters):
    # 对MSE求导
    res = np.reshape(np.dot(X, self.theta), (-1,))
    error = res - train_y
    update = [np.reshape(error * X[:, j], (-1, 1)) for j in range(len_theta)]
    update = np.hstack(update)
    update = np.reshape(np.mean(update, axis=0), (-1, 1))
    # 更新学习率（每隔一定的迭代次数就按比缩小学习率）
    if i > 0 and i % alpha_v_step == 0:
        alpha = alpha * alpha_v_rate
    # 更新参数（若含正则项，则会在梯度下降前适当缩小原系数）
    self.theta = self.theta * (1-alpha*(lamda/num_data)) - alpha*update
```

```
        losses.append(self.mse_loss(train_x, train_y))
```

在实现的线性回归算法上，分别进行下面四个实验。

1. 实现四种不同的线性回归模型训练方式：正规方程，正规方程+正则化，梯度下降，梯度下降+正则化。每种训练方法在myLinearRegression.py中都有实现，并使用K折交叉验证法对比四种训练方式的效果。

```
# 实验1：对比四种不同的模型训练方式（ [正规方程法; 梯度下降法] × [无正则化; 正则化] ）
def compare_training_method(model, x, y):
    """结论：在波士顿房价数据集下，正规方程法最快且效果好，正则化效果会不稳定"""
    methods, scores = ['ne', 'bgd', 'ne_reg', 'bgd_reg'], []
    for method in methods:
        score = cross_val_score(model, x, y, n_splits=5, fit_method=method,
image_show=False)
        scores.append(score)
    print('\n\nAverage Score (MSE_Loss, R2_score): ')
    print(dict(zip(methods, scores)))
```

2. 探讨了特征缩放feature scaling的作用（不进行缩放 / 进行标准化 / 进行归一化）

```
def feature_scaling_test(model, x, y):
    data1 = x
    data2 = MinMaxScaler().fit_transform(x)
    data3 = scale(x)
    # 正规方程法
    print('NE_TEST')
    for d in [data1, data2, data3]:
        print(cross_val_score(model, d, y, fit_method='ne', image_show=False))
    # 梯度下降法
    print('BGD-TEST')
    # 不进行特征缩放
    alphas = [1e-5, 1e-6, 1e-7]
    losses = []
    train_x, test_x, train_y, test_y = train_test_split(data1, y,
test_size=0.25, random_state=1)
    for alpha in alphas:
        losses.append(model.fit_bgd(train_x, train_y, alpha=alpha))
        ...;
    # 进行标准化、归一化
    alphas = [1e-2, 5e-3, 1e-3, 5e-4, 1e-4]
    ymaxs = [200, 250, 300, 450, 500]
    losses = []
    for i, alpha in enumerate(alphas):
        for d in [data2, data3]:
            train_x, test_x, train_y, test_y = train_test_split(d, y,
test_size=0.25, random_state=1)
            losses.append(model.fit_bgd(train_x, train_y, alpha=alpha))
        ...;
```

3. 绘制模型的学习曲线，探讨模型的拟合效果

```
def draw_learning_curve(model, x, y):
    train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.3,
random_state=3)
    num_of_train = range(1, len(train_x) + 1, 3)
```

```
    for n in range(2):
        ...; #略去绘图代码
        # 逐渐增大训练样本，查看模型在训练集和测试集上的MSE损失值
        for i in num_of_train:
            # 分别查看对正规方程法和梯度下降法的影响
            if n == 0:
                model.fit_ne(train_x[:i, :], train_y[:i])
            else:
                model.fit_bgd(train_x[:i, :], train_y[:i])
            train_scores.append(model.mse_loss(train_x[:i, :], train_y[:i]))
            test_scores.append(model.mse_loss(test_x, test_y))
            ...; #略去绘图代码
```

4. 探讨正则项系数大小对于模型的影响

```
def lambda_test(model, x, y):
    lamdas = list(range(50))
    train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.25,
random_state=7)
    for i in range(2):
        ...; #略去绘图代码
        for lamda in lamdas:
            # 分别查看正则项系数变化对正规方程法和梯度下降法的影响
            if i == 0:
                model.fit_ne(train_x, train_y, regularized=True, lamda=lamda)
            else:
                model.fit_bgd(train_x, train_y, regularized=True, lamda=lamda)
            train_scores.append(model.mse_loss(train_x, train_y))
            test_scores.append(model.mse_loss(test_x, test_y))
            ...; #略去绘图代码
```

具体实验过程见**学习率与正则化参数等分析。**

**②线性回归模型的高维扩展与实现（刘旭东）**

上述计算方法只限于二维的线性回归算法，而对于高维特征数据的线性回归算法，需要进行更高维度数据的运算与求导。求导的运算方式与上面方法类似，但对于更高维的数据，求导速度往往会受维度增加而下降，因此为保证运算速度以及准确度，这里使用pytorch的自动求导方法来代替上述的人工求导。除求导以外的内容均自行编写。

这里将扩展到高维的线性回归算法封装为一个LinearRegression类。初始化函数如下。其中epoch为训练轮次，lr为学习率，lambda_l1、lambda_l2分别为使用L1正则项或L2正则项的正则项系数。

```
class LinearRegression():
    def __init__(self,epoch=500,lr=0.01,lambda_l1=0.01,lambda_l2=0.01):
        self.lr=lr
        self.epoch=epoch
        self.lambda_l1=lambda_l1
        self.lambda_l2=lambda_l2
```

类中封装了类似于sklearn的训练函数，如fit、predict、score、get_params等。另外，为了应对输入数据量过高的情况，加入fit_MBGD函数，用于进行mini batch的梯度下降算法。

```
class LinearRegression():
    ...;
    def fit_MBGD(self,X,y,batch_size=50,regular='l2',print_process=True):
```

```
        X_len=X.shape[0]
        self.w=torch.rand([X.shape[1],1],requires_grad=True)#初始化参数
        self.b=torch.rand(1,requires_grad=True)

        for _ in range(self.epoch):
            shuffle_indices=np.random.permutation(X_len)#随机排列
            X_shuffle=X[shuffle_indices]
            y_shuffle=y[shuffle_indices]
            for i in range(0,X_len,batch_size):
                xi=X_shuffle[i:i+batch_size]#选取mini batch进行训练
                yi=y_shuffle[i:i+batch_size]
                yi_pred=torch.matmul(xi,self.w)+self.b
                lossi=(yi_pred-yi).pow(2).mean()
                #损失函数加正则项
                if regular=='l2':
                    lossi+=(self.w.pow(2).sum()+self.b.pow(2)[0])*self.lambda_l2
                elif regular=='l1':
                    lossi+=(self.w.abs().sum()+self.b.abs()[0])*self.lambda_l1

                loss_recorder.append(lossi.detach().numpy())#记录损失

                if self.w.grad is not None:
                    self.w.grad.data.zero_()
                if self.b.grad is not None:
                    self.b.grad.data.zero_()
                lossi.backward()

                if print_process:
                    print("epoch {}/{}, batch {}/{}, loss:
{}".format(_,self.epoch,i,X_len,lossi))
                self.w.data-=self.lr*self.w.grad.data #更新参数
                self.b.data-=self.lr*self.b.grad.data
```

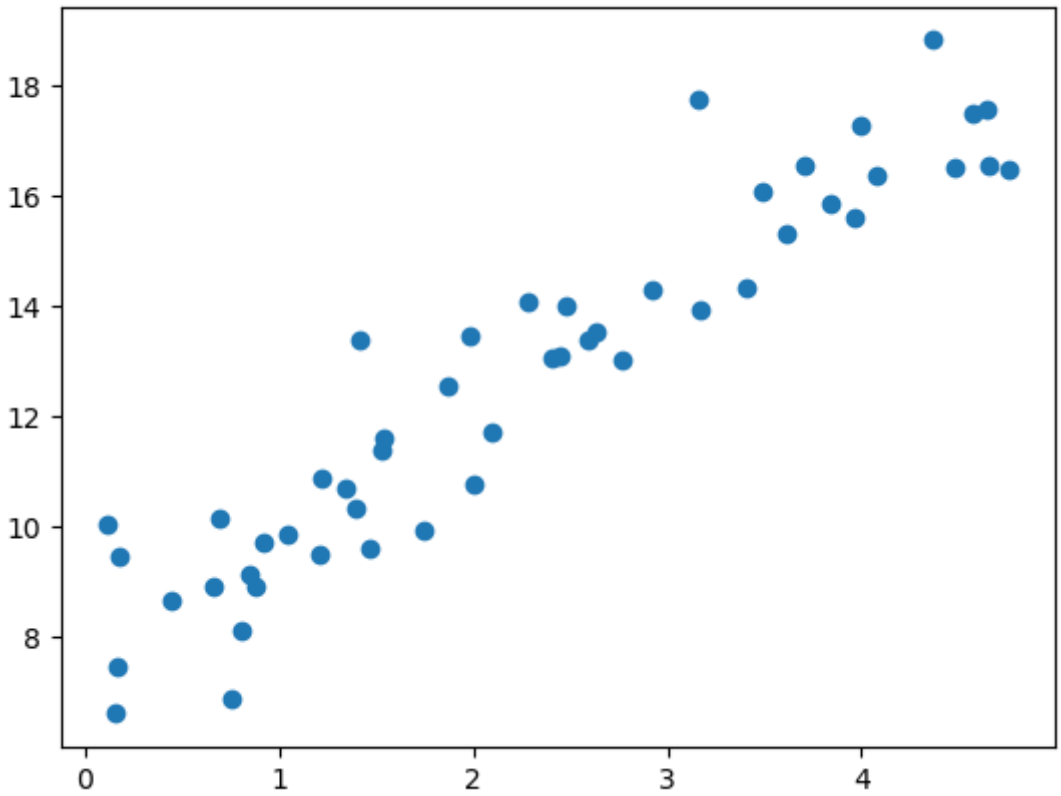
接下来进行线性回归类的实验。包括对于二维输入数据以及多维输入数据的实验。

首先生成模拟回归数据。

```
#二维模拟回归数据
x = torch.rand([5000,1])*5
y_true = torch.matmul(x,torch.Tensor([[2]]).T) + 8
y=y_true+torch.randn(y_true.shape)
#查看数据
plt.scatter(x.numpy(),y.numpy())
plt.show()

#高维模拟回归数据
x = torch.rand([5000,5])*5
y_true = torch.matmul(x,torch.Tensor([[1,2,3,4,5]]).T) + 8 #按照计算公式y的准确值
y=y_true+torch.randn(y_true.shape)*2
```

生成的数据如下所示。




在生成数据上进行训练

```
#拟合
model=LinearRegression(epoch=500,lr=0.01,lambda_l2=0.01)
model.fit(x,y,regular='l2')
y_pred=model.predict(x)
visualization_for_predict_data(y.numpy(),y_pred.detach().numpy(),'y_train','y_pr
ed')
#一维数据的数据点可视化以及拟合直线可视化
plt.scatter(x.numpy(),y.numpy())
w,b=model.get_params()
plt.plot([x.min(),x.max()],
[np.matmul([x.min()],w)+b,np.matmul([x.max()],w)+b],'r')
plt.show()
#查看损失函数曲线
plt.plot(np.arange(len(loss_recorder)),loss_recorder)
plt.show()
#进行k折分析，并绘制每一折的图像
regression(model,x,y)
```

## 2.4 其他创新内容或优化算法

1. 在二维线性回归类中加入类似于sklearn的接口，并加入多种训练方式，包括未归一化的梯度下降训练，归一化的梯度下降训练，未归一化的正规方程法训练，归一化的正规方程法训练，将所有训练方法整合到一个fit函数中，并通过参数对训练方法进行调整。

```
def fit(self, train_x, train_y, method=2):
    """快速进入不同的训练方法"""
    if method in [1, 'ne']:
        self.fit_ne(train_x, train_y)
    elif method in [2, 'bgd']:
        self.fit_bgd(train_x, train_y)
    elif method in [3, 'ne_reg']:
        self.fit_ne(train_x, train_y, regularized=True)
    elif method in [4, 'bgd_reg']:
        self.fit_bgd(train_x, train_y, regularized=True)
```

2. 将正则化方式作为参数在fit函数中导入，可在每次训练时选择不同的正则化方式，使模型训练更灵活。

```
class LinearRegression():
    def __init__(self,epoch=5000,lr=0.01,lambda_l1=0.01,lambda_l2=0.01):
        ...;
    def fit(self,X,y,regular='l2',print_process=True):
        ...;
            if regular=='l2': #L2正则化
                loss+=(self.w.pow(2).sum()+self.b.pow(2)[0])*self.lambda_l2
            elif regular=='l1': #L1正则化
                loss+=(self.w.abs().sum()+self.b.abs()[0])*self.lambda_l1
```

3. 在全数据集训练函数之外添加一个能够以mini batch方式训练的函数，方便直接通过调用类函数的方法进行不同训练方式的对比。并以参数方式传入batch大小、正则化方法等。

```
class LinearRegression():
    def fit(self,X,y,regular='l2',print_process=True):
        ...;
    def fit_MBGD(self,X,y,batch_size=50,regular='l2',print_process=True):
        ...;
```

# 3 实验过程

## 3.1 环境说明

```
## 刘旭东
操作系统：Windows 10
开发语言：python
开发环境与版本：Anaconda 4.8.2 python 3.7.7
核心使用库：
scikit-learn     0.22.1
numpy            1.21.3
pandas           1.0.3
matplotlib       3.1.3
torch            1.9.0

##王攀栋
操作系统：Windows 10
开发语言：python
开发环境与版本：Anaconda 4.8.2 python 3.7
核心使用库：
scikit-learn     0.22.1
```

```
numpy            1.21.3
matplotlib       3.1.3

## 王国润
操作系统：macOS
开发语言：python
开发环境与版本：Anaconda 4.8.2 python 3.9
核心使用库：
seaborn          0.11.2
scikit-learn     1.0.2
statsmodels      0.13.2
matplotlib       3.5.1
```

## 3.2 源代码文件清单

```
## 刘旭东
- sklearn_regression.py '调用scikit-learn进行回归训练'
    - train_test_split(X,y,test_size=0.2,random_state=27) #将数据集划分为训练集与测试集
    - MinMaxScaler() #将输入数据进行归一化
    - train_score(regr,X_train,y_train) #拟合输入数据，并进行k折交叉验证
    - regression(model,data,target,splits=5,size=0.2) #自定义k折交叉验证，可查看每次交叉验证后的预测图像，并计算每折的决定系数
    - plot_predict_label(y,y_pred) #绘制预测值与真实值的分布，查看预测效果
    - visualization_for_predict_data(data_observed,data_predicted,label_observed,label_predicted) #以横纵坐标值的形式绘制预测值与真实值，查看预测效果
    - test_measure_visualization(X,y,regr) #传入训练好的模型并进行测试集预测与可视化
- regression.py '自定义回归模型'
    - class LinearRegression() #自定义线性回归类
        - __init__(self,epoch=5000,lr=0.01,lambda_l1=0.01,lambda_l2=0.01) #初始化回归对象，传入训练epoch、学习率、正则化参数等
        - fit(self,X,y,regular='l2',print_process=True) #训练函数，输入训练数据与标签，可选择正则化方式，以及是否打印训练过程
        - predict(self,X) #预测函数，输入测试数据返回预测值
        - score(self,X,y) #评分函数，输入测试数据与标签，函数中计算R2score作为分数返回
        - get_params(self) #返回模型参数
        - fit_MBGD(self,X,y,batch_size=50,regular='l2',print_process=True) #以mini batch的方式进行训练，参数中可选择一次训练的batch大小、正则化方式等
    - visualization_for_predict_data(data_observed,data_predicted,label_observed,label_predicted) #以横纵坐标值的形式绘制预测值与真实值，查看预测效果
    - regression(model,data,target,splits=5,size=0.2) #自定义k折交叉验证，可查看每次交叉验证后的预测图像，并计算每折的决定系数

##王攀栋
- myLinearRegression.py '含线性回归模型MyLinearRegression的Class构造'
    - fit #训练函数（具体包含两种训练方式：fit_ne正规方程法，fit_bgd梯度下降法）
    - predict #预测函数
    - mse_loss #MSE损失函数
    - score #R2系数评价函数
- myLR_test.py '含相关测试实验'
    - cross_val_score #K折交叉验证函数，可返回模型在每折训练集上的平均MSE与R2
```

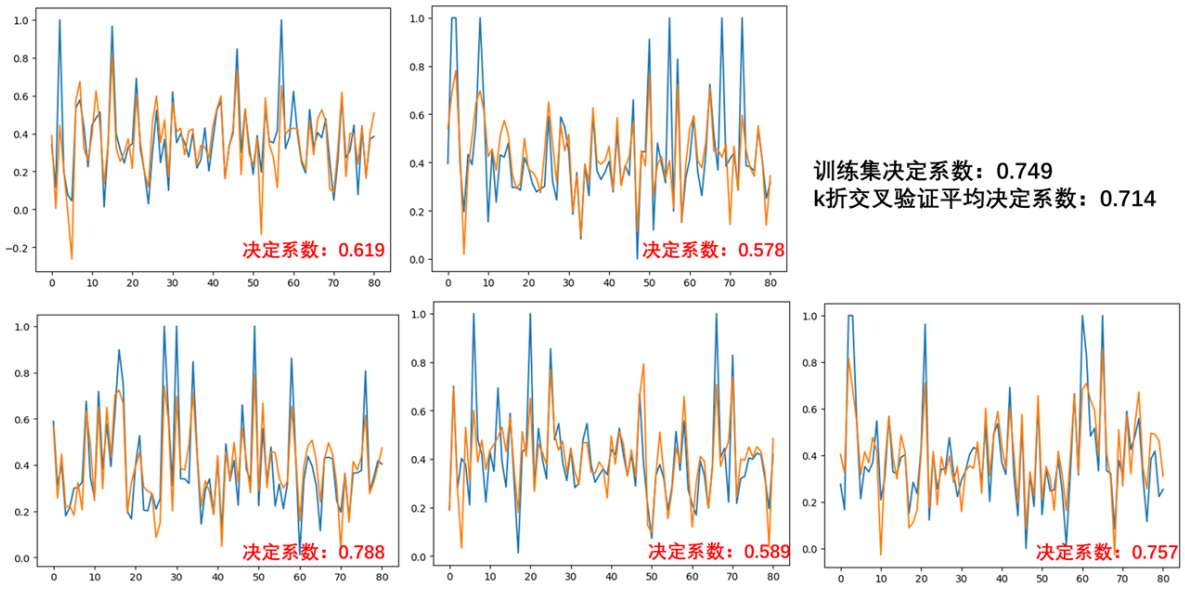
```
    - compare_training_method #（实验1）利用K折交叉验证法，对比四种不同的模型训练方式（分别是正规方程法，梯度下降法，正则化正规方程法，正则化梯度下降法）
    - feature_scaling_test #（实验2）探究特征缩放的作用，通过调节学习率，对比三种训练数据处理方式（分别是不进行特征缩放、MinMax缩放、标准化）
    - draw_learning_curve #（实验3）调节训练样本数，绘制模型的学习曲线，探究模型的拟合效果
    - lambda_test #（实验4）探讨正则项系数对模型训练的影响

## 王国润
- main.py
- main2.py
- main4.py
    - StandardScaler() #数据标准化
    - train_test_split() #切分训练数据与测试数据
    - r2_score() #计算R2 score
    - mean_squared_error() #计算均方误差
    - LinearRegression() #线性回归
```

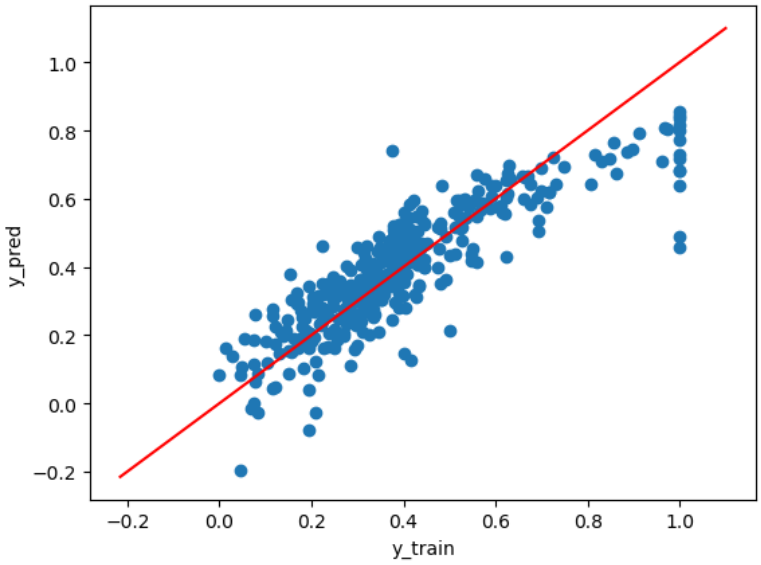
## 3.3 实验结果展示

### 1 通过scikit-learn模块进行线性回归实验（刘旭东）

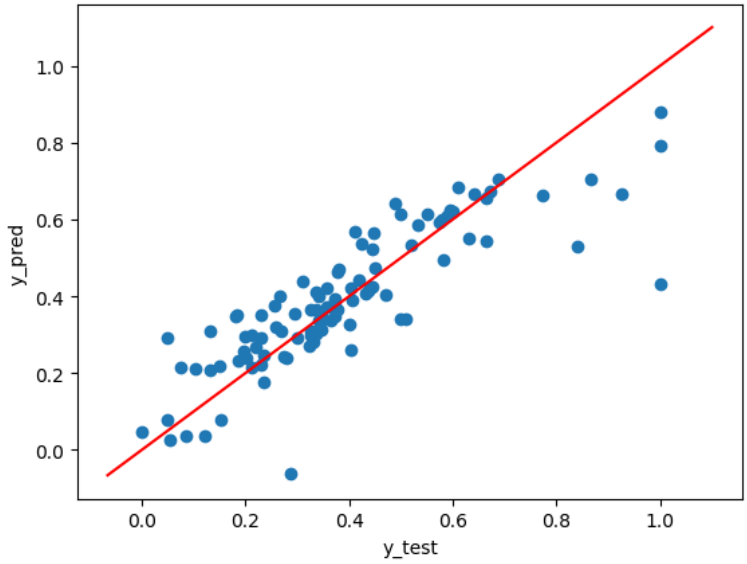
通过上述两种交叉验证方法得到的预测值与观测值对比图，以及计算的决定系数如下。



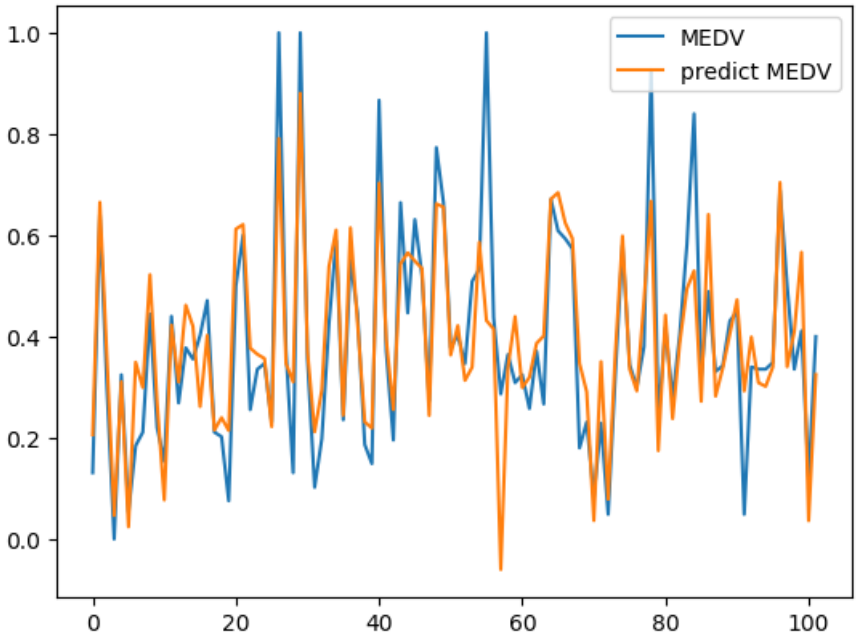

训练集通过visualization_for_predict_data得到的散点图如下图。可见大部分样本点的预测都集中在斜率为1的直线附近，说明预测较为准确。




测试集通过visualization_for_predict_data得到的散点图如下图。图中除了少部分离群样本点外，其余样本点基本都分布在斜率为1的直线附近，在测试集的回归效果也较为理想。




测试集的样本观测值与预测值的对比图如下，可见有若干样本点预测偏差较大，但大部分预测较为准确，与上述散点图情况相同。



## 2 数据进一步分析，学习率与正则项（王国润、王攀栋）

**①实验1：对比四种不同的模型训练方式**

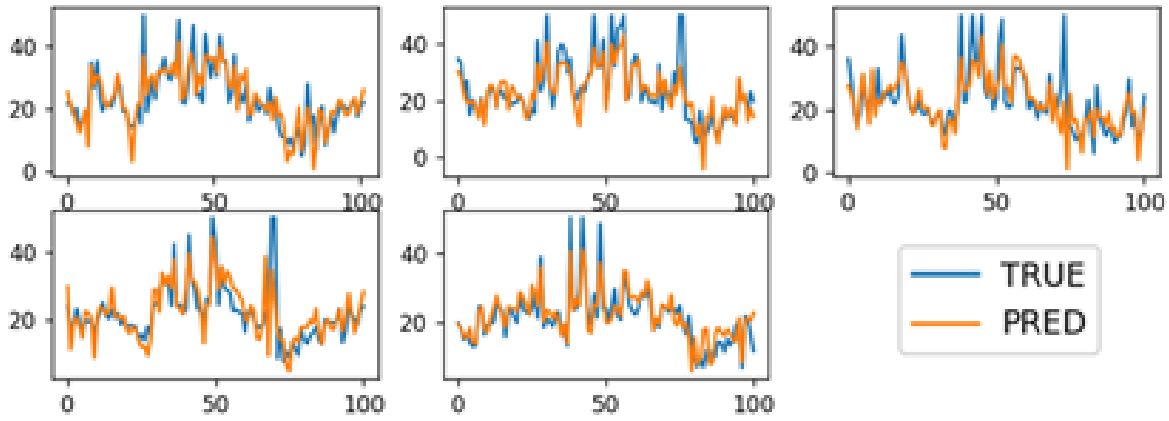
利用K折交叉验证法进行评价。取K=5，输出每一折在测试集上的MSE损失值与R2评价值，并取平均，结果如下表。（训练参数详见MyLinearRegression类中训练函数的默认值）

| 训练方法 | 每折测试集的MSE损失值 | | | | | 平均MSE |
|---|---|---|---|---|---|---|
| NE | 18.495 | 28.340 | 31.224 | 24.908 | 13.530 | 23.299 |
| BGD | 18.495 | 28.339 | 31.226 | 24.911 | 13.531 | 23.300 |
| reg-NE | 18.786 | 31.333 | 31.105 | 23.337 | 14.096 | 23.731 |
| reg-BGD | 19.472 | 31.231 | 31.631 | 24.379 | 14.294 | 24.201 |

| 训练方法 | 每折测试集的R2评价值 | | | | | 平均R2 |
|---|---|---|---|---|---|---|
| NE | 0.7789 | 0.7541 | 0.6646 | 0.6275 | 0.7723 | 0.7195 |
| BGD | 0.7789 | 0.7541 | 0.6646 | 0.6275 | 0.7723 | 0.7195 |
| reg-NE | 0.7755 | 0.7281 | 0.6659 | 0.6510 | 0.7628 | 0.7167 |
| reg-BGD | 0.7672 | 0.7290 | 0.6602 | 0.6354 | 0.7594 | 0.7102 |

由上表可知，**在充分迭代后，梯度下降法与正规方程法模型效果很接近，但在当前数据规模下，正规方程法更快。**此外，引入正则化后两种方法效果都略有下降，可能是正则项系数的选择不当，将在实验3中具体探讨。

此外，为直观了解模型的预测准确性，绘制模型在每一折测试集上的预测值与真实值，下图为梯度下降法的输出结果（初始学习率为0.1，迭代20000次）。




由上图得，**模型预测值大致接近真实值，但对一些极大值无法很好地拟合。猜测房价可能与某些特征有非线性关系，因此线性模型无法很好地预测出这些特殊情况。**

**②实验2：探究特征缩放与学习率的作用**

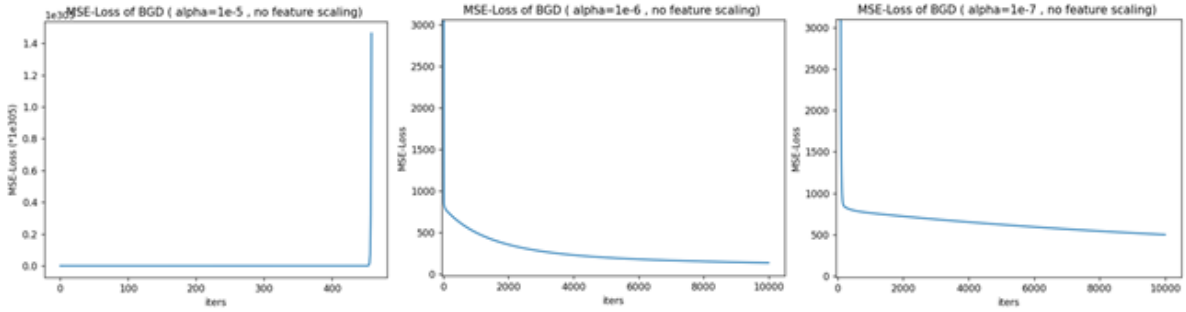
**对于正规方程法，不同的数据处理方式不影响矩阵计算结果**，如下表所示，因此不需要特征缩放。

| 数据处理 | NE：每折测试集的MSE损失值 | | | | | 平均MSE |
|---|---|---|---|---|---|---|
| 原始数据 | 18.495 | 28.340 | 31.224 | 24.908 | 13.530 | 23.299 |
| MinMax | 18.495 | 28.340 | 31.224 | 24.908 | 13.530 | 23.299 |
| Standard | 18.495 | 28.340 | 31.224 | 24.908 | 13.530 | 23.299 |

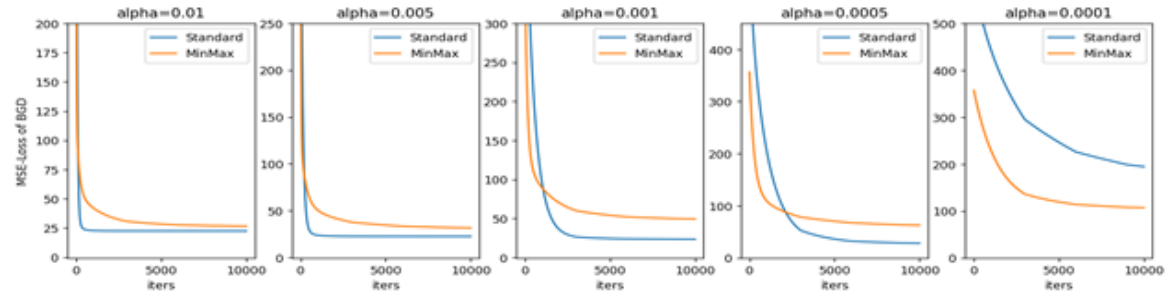
对于梯度下降法，则需要先考虑初始学习率的设置。

若使用原始数据集，由于不同特征数值范围不同，选择合适的初始学习率将变得很困难，很可能导致模型出现MSE损失值急速增加或极缓下降的情况（可理解为'步子'太大或太小）。在下图中，α取1e-5时，损失值一直在上升，迭代到500次之前就超过了浮点数上限；取1e-6时，损失值下降相对正常，但收敛很慢；取1e-7时，收敛速度更加不可行。**因此，很难找到最佳的学习率，即便找到其收敛速度也难以提**

**高**（可以理解为很难统一不同系数的更新调整步调）。如下图为梯度下降法在原始数据集上的表现（学习率分别为1e-5, 1e-6, 1e-7）。

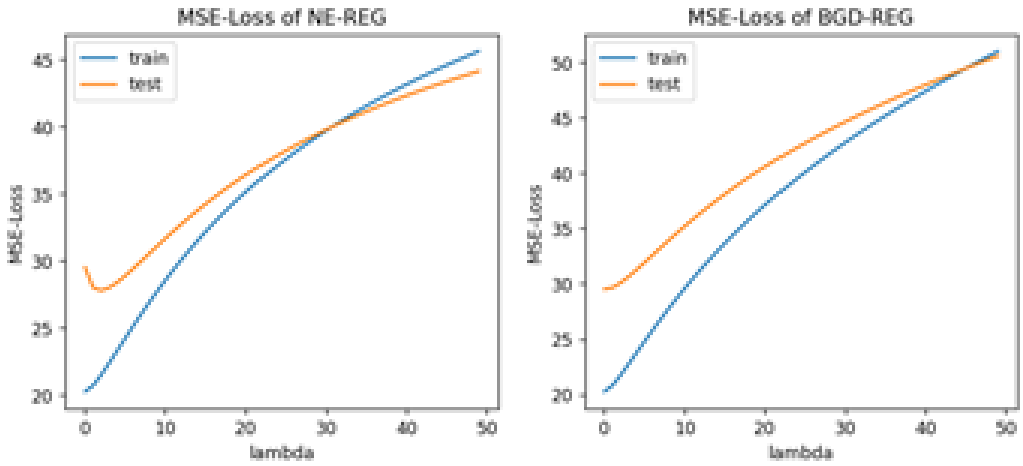



若进行MinMax缩放或Standard标准化，初始学习率的可考虑范围会变得更大，如下图所示。随着初始学习率的减小，损失值在迭代时的收敛速度也在变缓。两种数据处理方式的收敛特点也不同，MinMax在最开始迭代时下降速率要快于Standard，但此后下降速度就不如后者。**总体来说，Standard收敛效果比较好，初始学习率在[0.1, 0.01]区间内选取较好。**如下图为在不同初始学习率下，梯度下降法在MinMax缩放和Standard标准化时的表现。




**③实验3：探讨正则项系数对模型训练的影响**

正规方程和梯度下降在不同正则项系数λ下训练集与测试集的损失值变化如下图所示。




由图可知，随着λ的增大，对回归模型系数的抑制作用变大，模型对训练集的拟合效果越来越差，而对测试集的损失值会先有小幅下降（但有时也没有明显的下降过程，这与数据有关），然而才会逐渐升高。说明λ的取值有一个最佳值，需要进一步细化搜索。**在当前情景下，λ在区间[1，2]内取值时模型综合效果相对较好。**

## 3 数据集的数据变换（相关性、升维降维）与实验（王国润）

通过上述分类器的summary进行判断，筛掉了第三和第七个特征。而值得注意的是，这个正好为上文热力图分析的某些有相关性的特征有关联。训练完成后计算的决定系数也从0.6354变为0.6369，有一部分的提高，但综合来看分数还是很低，因此可以探究升维方法。

通过上述升维方法得到的数据进行训练与测试，测试得到的决定系数提升至0.7548。维度变换如下图所示。

```
print(pf.n_input_features_)
print(pf.n_output_features_)
```

```
/Users/caffrey/miniforge3/envs/tongji/lib/python3.9/si
  warnings.warn(msg, category=FutureWarning)
11
78
```

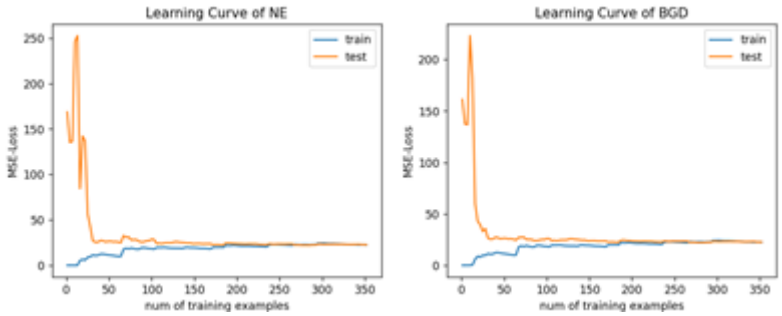
## 4 线性回归算法实现（王攀栋、刘旭东）

### ①二维线性回归模型实现（王攀栋）

在二维线性回归模型上进行的四个实验在**数据进一步分析，学习率与正则项**部分已经展示了三个，这里只展示最后一项实验。

**实验4：绘制模型的学习曲线，探究模型拟合效果**

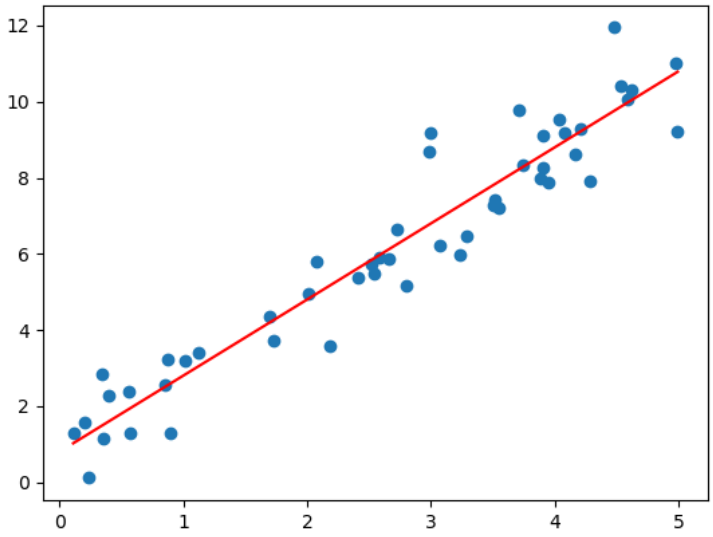
划分得到训练集与测试集（7:3）后，调整训练样本数，重复训练模型（分别使用正规方程和梯度下降），记录每个模型对其训练样本和测试集的预测效果，绘制得到模型的学习曲线，如下图所示为正规方程法与梯度下降法的学习曲线。




在图中，随着训练样本数的增加，训练集损失值先上升后稳定，测试集先下降后与训练集曲线重合（可以理解为最开始训练样本少，模型很容易贴合训练集，而难以判断未知的测试集，而之后模型趋于稳定）。这就说明，**模型在整个数据集上达到了相对稳定状态，没有必要再增加训练集。**但考虑到上图中无法预测出极大值的情况，说明此时**模型可能处于高偏差（欠拟合）状态**，之后可以考虑引入非线性关系。

### ②线性回归模型高维扩展（刘旭东）

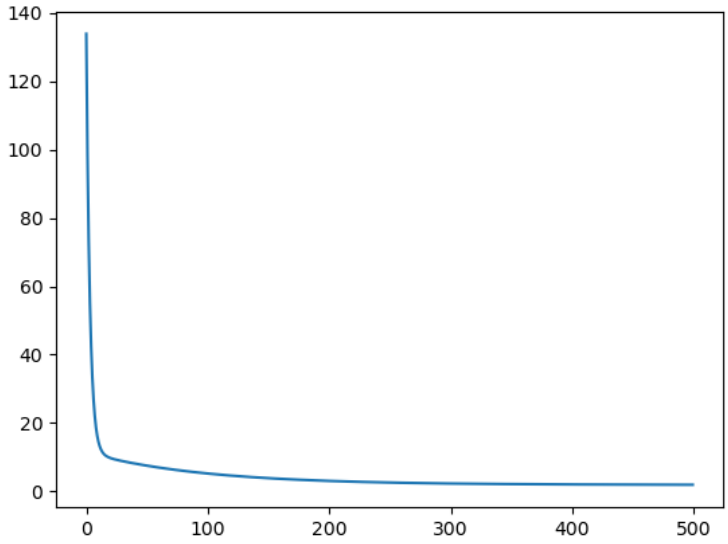
二维数据回归效果，模型能够实现较为准确的回归。



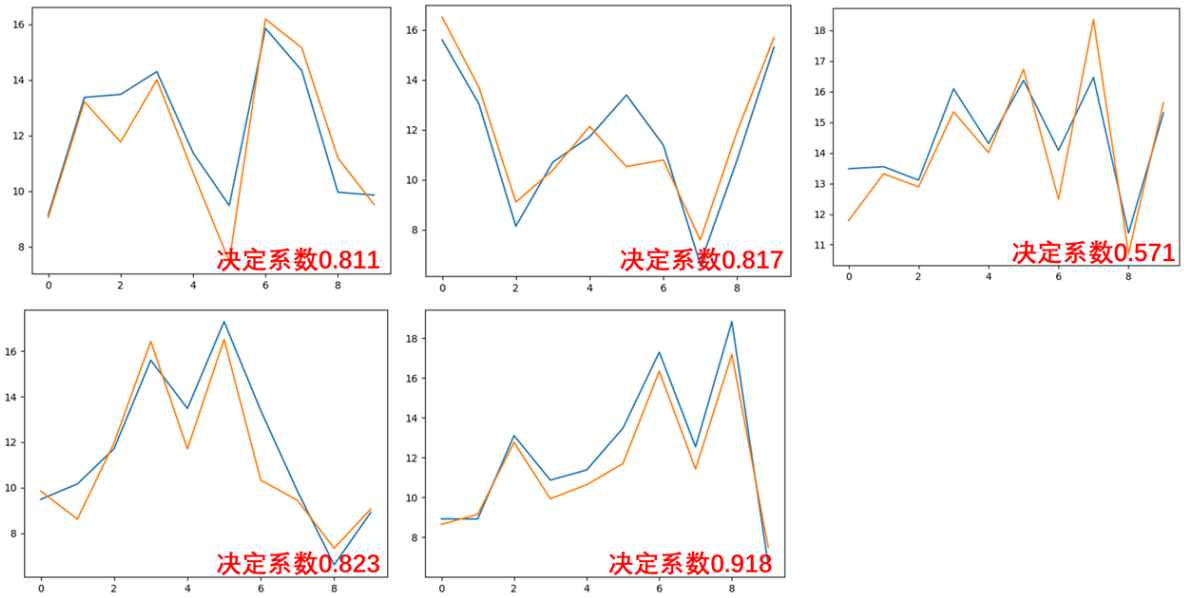

二维数据k折，每一轮计算的决定系数以及平均决定系数都较高。

```
第4折 决定系数 0.777
fold 5/5
第5折 决定系数 0.800
平均决定系数  0.8398239083536796
```

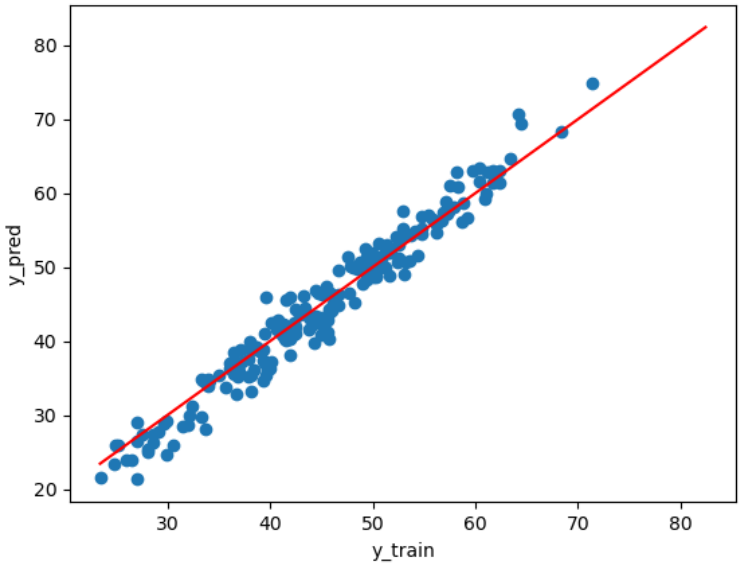
损失函数下降曲线




k折中每折预测值与观测值对比与计算的决定系数



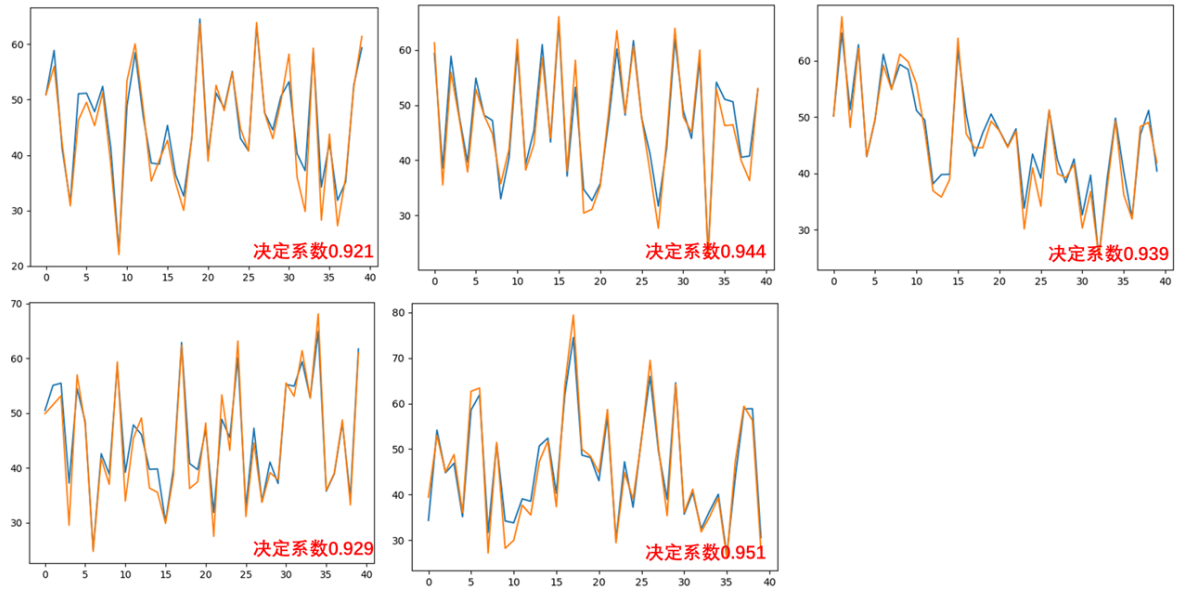

高位数据预测值与观测值对比。




高维数据k折，每一轮计算的决定系数以及平均决定系数都较高。

```
fold 2/5
第2折 决定系数 0.972
fold 3/5
第3折 决定系数 0.934
fold 4/5
第4折 决定系数 0.852
fold 5/5
第5折 决定系数 0.970
平均决定系数  0.8995025655918898
```

k折中每折预测值与观测值对比与计算的决定系数，可见模型对于高维数据的拟合能力更强。




## 3.4 实验结论

通过k折交叉验证能够更加客观反应模型效果，一定程度防止模型过拟合。

正则化能够加强模型扩展性，防止模型过于复杂。不同正则化方法对于训练效果有一定影响。一般使用L2正则项。

通过数据升维与降维能够一定程度提升模型准确率，数据升维效果要略强于数据降维。

手写的线性回归算法较适用于本身含有较高线性相关性的数据，并且对于高维数据拟合能力更强。

mini batch梯度下降算法能够加快大量数据的训练速度。

# 4 总结

## 4.1 实验中存在的问题与解决方案

①损失函数中使用均方误差训练效果较差，容易受真实值数量级的影响。

解决方案：使用决定系数衡量模型损失，能够消除真实值数量级的影响。

②numpy的矩阵与向量计算问题

初期发现模型的梯度下降训练非常缓慢，且迭代过程中损失值一直在增大，正规方程法的损失值也异常的大。最后发现是在计算损失值时，真实值与预测值分别是numpy向量和矩阵，它们的shape分别是(404, )和(404, 1)，在计算MSE时numpy会自动广播形成(404,404)的庞大矩阵，导致损失值计算缓慢且结果异常。

解决方案：确保真实值与预测值均是numpy的向量

## 4.2 心得体会

刘旭东：本次实验较好地完成了老师的实验要求，实验效果达到了理想效果。后续还可以在参数初始化方法、学习率自动调整、训练轮次自动调整等方面进行完善。

王国润：在进行调用库的时候应当查看api源码，深刻体会内部的关系后再使用。

王攀栋：在使用numpy时，需要时刻注意变量的shape，不要混淆向量与矩阵，避免numpy的广播机制带来困扰。在实际操作时，可以多加一些reshape函数，进一步提升程序的可用性。在设计实验时，需要把握好"先粗调后微调"的思想，避免将变量的取值局限在某一个小范围内，否则实验结果可能会同时出现异常，这个时候就不好区分是代码问题还是数值问题了。

## 4.3 后续改进方向

**①参数初始化方法**

在自定义的线性回归模型中，参数初始化方法为按照正态分布的随机初始化。这种方法虽然常用但并不总是最有效的。一个好的参数初始化方法往往能够提高模型训练的效率与效果。

还可以选择的参数初始化方法有：

- Xavier正态分布初始化器

  它从以0为中心，标准差为$\sqrt{\frac{2}{fan_{in}+fan_{out}}}$的截断正态分布中抽取样本， 其中$fan_{in}$是权值张量中的输入单位的数量， $fan_{out}$是权值张量中的输出单位的数量。

- 正态化的kaiming初始

  He正态分布初始化器。它从以0为中心，标准差为$\sqrt{\frac{2}{fan_{in}}}$的截断正态分布中抽取样本， 其中$fan_{in}$是权值张量中的输入单位的数量。

- LeCun 均匀初始化器

  它从$[-limit，limit]$中的均匀分布中抽取样本， 其中$limit$是$\sqrt{\frac{3}{fan_{in}}}$， $fan_{in}$是权值张量中的输入单位的数量。

**②学习率自动调整**

目前模型中的学习率是固定的。学习率过大可能导致模型效果变差，过小可能导致模型收敛速度较慢。但如果学习率一开始输入较大，当训练进行到某一程度时减小学习率，就可以兼顾大学习率学习速度快以及小学习率训练精度高等优点。

```
lr=0.1
SHRINK=100
for i in range(epoch):
    ...;
    if(!(i%SHRINK)):
        lr/=10
    ...;
```

**③训练轮次自动调整**

训练轮次过少，模型可能仍未收敛，训练轮次过高又会浪费训练时间。因此可以加入一个判断模型是否收敛的函数来控制训练轮次，当模型达到收敛后就停止训练，节约训练时间。

```
def early_break(loss_recorder,loss,thres):
    if abs(loss-loss_recorder[-1])<thres:
        break
```

④手动实现sklearn中的所有函数，更加深刻的体会各个参数的关系，并讨论各个因素对模型整体对影响。

**⑤尝试更高维的回归模型**

实验中发现，线性模型难以准确预测出某些较高的房价值，说明线性回归模型比较适合中低房价的预测，而高房价与数据特征之间存在某些非线性关系。因此可以尝试构造出13个特征的二次项甚至三次项，根据实验结果保留关键的二次项，提高模型对这种非线性关系的拟合能力（要注意需重新调整正则项系数）。另外，可以尝试其他的优化训练方法，比如共轭梯度法、拟牛顿法。

## 4.4 总结

本次实验在波士顿房价数据集以及修正后的波士顿房价数据集上进行了线性回归预测。首先通过sklearn的LinearRegression类进行模型训练，并测试训练效果，同时进行超参数的调整，包括学习率、正则项等来优化模型训练效果。同时进行数据的进一步分析，包括数据特征之间的相关性、数据升维与数据降维等，模型预测准确率有了进一步的提高。

同时，我们自己实现了线性回归算法，并将输入数据的维度扩展到了高维，并进行了代码实现。将线性回归各种功能封装为一个类。同时加入mini batch随机梯度下降算法来应对输入数据量很大的情况。模型在线性回归数据上进行实验并取得了较为良好的效果。

# 5 参考文献


[1] Harrison, David, and Daniel L. Rubinfeld, Hedonic Housing Prices and the Demand for Clean Air, Journal of Environmental Economics and Management, Volume 5, (1978), 81-102. Original data.

[2] Gilley, O.W., and R. Kelley Pace, On the Harrison and Rubinfeld Data, Journal of Environmental Economics and Management, 31 (1996), 403-405. Provided corrections and examined censoring.

[3] Pace, R. Kelley, and O.W. Gilley, Using the Spatial Configuration of the Data to Improve Estimation, Journal of the Real Estate Finance and Economics 14 (1997), 333-340.

[4] 周志华，机器学习，清华大学出版社，2016

[5] scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation. (2022). Retrieved 14 March 2022, from https://scikit-learn.org/stable/

[6] PyTorch. (2022). Retrieved 14 March 2022, from https://pytorch.org/


# 6 成员分工与自评

## 6.1 成员分工

刘旭东：

1. 通过scikit-learn模块进行线性回归实验
2. 线性回归模型的高维扩展与实现
3. 整合实验报告

王攀栋：

1. 进行训练过程中的学习率与正则化参数分析
2. 进行二维线性回归算法实现
3. 撰写实验报告

王国润：

1. 进行数据集的数据变换（相关性、升维降维）与实验
2. 撰写实验报告

## 6.2 自评

刘旭东：本次实验较好地完成了老师的实验要求，实验效果达到了理想效果。在编写线性回归算法部分通过torch库与mini batch SGD优化高维向量与大量数据在训练时的效率与速度，并达到了较为理想的效果。后续还可以在参数初始化方法、学习率自动调整、训练轮次自动调整等方面进行完善。

王国润：积极查阅资料，探究问题，但未手动实现代码，还要继续努力拼。

王攀栋：由于没有提前设计好整体的实验步骤，导致写代码的时候一直在东改西凑，函数调来调去乱成一团。以后要养成先画逻辑图的习惯，明确好每一个函数的具体功能，不能急急忙忙直接上手代码。