## 2.1.5 公共健康数据的线性回归与梯度下降法[1]

公共健康数据可以分为三类，分别为地理信息数据、疾病数据和经济条件数据。在公共健康数据中，将经济条件数据视为自变量，疾病数据视为因变量。通常自变量为因，因变量为果。自变量是可以改变的因素，而因变量为不能改变的因素。 公共健康数据中英对照表（字段映射表）：

PubicHealth\_Statistic\_columns={'geographic information':  
 {'Community Area':'社区',   
 'Community Area Name':'社区名',},  
 'disease':  
 {'natality':{'Birth Rate':'出生率',  
 'General Fertility Rate':'一般生育率',  
 'Low Birth Weight':'低出生体重',  
 'Prenatal Care Beginning in First Trimester':'产前3个月护理',   
 'Preterm Births':'早产',  
 'Teen Birth Rate':'青少年生育率',},  
 'mortality':{'Assault (Homicide)':'攻击（杀人）',  
 'Breast cancer in females':'女性乳腺癌',  
 'Cancer (All Sites)':'癌症',   
 'Colorectal Cancer':'结肠直肠癌',  
 'Diabetes-related':'糖尿病相关',  
 'Firearm-related':'枪支相关',  
 'Infant Mortality Rate':'婴儿死亡率',   
 'Lung Cancer':'肺癌',  
 'Prostate Cancer in Males':'男性前列腺癌',  
 'Stroke (Cerebrovascular Disease)':'中风(脑血管疾病)',},  
 'lead':{'Childhood Blood Lead Level Screening':'儿童血铅水平检查',  
 'Childhood Lead Poisoning':'儿童铅中毒',},  
 'infectious':{'Gonorrhea in Females':'女性淋病',   
 'Gonorrhea in Males':'男性淋病',   
 'Tuberculosis':'肺结核',},  
 'economic condition':  
 {'Below Poverty Level':'贫困水平以下',   
 'Crowded Housing':'拥挤的住房',   
 'Dependency':'依赖',  
 'No High School Diploma':'没有高中文凭',   
 'Per Capita Income':'人均收入',  
 'Unemployment':'失业',},  
 }

在实际的回归模型应用中，使用[scikit-learn（sklearn)](https://scikit-learn.org/stable/index.html)①库为主。对于有些描述也做相应的变化，自变量为解释变量（explanatory variable）即特征（features）或属性（attributes），其值为特征向量，通常用表示特征向量（vector）数组（array）或矩阵；因变量为响应变量（response variable），其值通常用表示（为目标值，target value）；机器学习（machine learning）的算法（algorithms）和模型（models），被称为估计器（estimator），不过算法、模型和估计器的叫法有时会混淆使用。

首先读取公共健康数据为DataFrame数据格式，因为sklearn的数据处理过程，以NumPy的数组形式为主，需要在二者之间进行转换。

通常用大写字母表示矩阵（大于或等于2维的数组），用小写字母表示向量。

import pandas as pd  
import geopandas as gpd  
  
dataFp\_dic={  
 "ublic\_Health\_Statistics\_byCommunityArea\_fp":r'./data/Public\_Health\_Statistics\_Selected\_public\_health\_indicators\_by\_Chicago\_community\_area.csv',  
 "Boundaries\_Community\_Areas\_current":r'./data/ChicagoCommunityAreas/ChicagoCommunityAreas.shp',   
 }  
  
pubicHealth\_Statistic=pd.read\_csv(dataFp\_dic["ublic\_Health\_Statistics\_byCommunityArea\_fp"])  
community\_area=gpd.read\_file(dataFp\_dic["Boundaries\_Community\_Areas\_current"])  
community\_area.area\_numbe=community\_area.area\_numbe.astype('int64')  
pubicHealth\_gpd=community\_area.merge(pubicHealth\_Statistic,left\_on='area\_numbe', right\_on='Community Area')  
  
pubicHealth\_gpd.head()

|  | **area** | **area\_num\_1** | **area\_numbe** | **comarea** | **comarea\_id** | **community** | **perimeter** | **shape\_area** | **shape\_len** | **geometry** | **...** | **Childhood Lead Poisoning** | **Gonorrhea in Females** | **Gonorrhea in Males** | **Tuberculosis** | **Below Poverty Level** | **Crowded Housing** | **Dependency** | **No High School Diploma** | **Per Capita Income** | **Unemployment** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.0 | 35 | 35 | 0.0 | 0.0 | DOUGLAS | 0.0 | 4.600462e+07 | 31027.054510 | POLYGON ((-87.60914 41.84469, -87.60915 41.844... | ... | 0.0 | 1063.3 | 727.4 | 4.2 | 26.1 | 1.6 | 31.0 | 16.9 | 23098 | 16.7 |
| **1** | 0.0 | 36 | 36 | 0.0 | 0.0 | OAKLAND | 0.0 | 1.691396e+07 | 19565.506153 | POLYGON ((-87.59215 41.81693, -87.59231 41.816... | ... | 0.3 | 1655.4 | 1629.3 | 6.7 | 38.1 | 3.5 | 40.5 | 17.6 | 19312 | 26.6 |
| **2** | 0.0 | 37 | 37 | 0.0 | 0.0 | FULLER PARK | 0.0 | 1.991670e+07 | 25339.089750 | POLYGON ((-87.62880 41.80189, -87.62879 41.801... | ... | 2.5 | 1061.9 | 1556.4 | 0.0 | 55.5 | 4.5 | 38.2 | 33.7 | 9016 | 40.0 |
| **3** | 0.0 | 38 | 38 | 0.0 | 0.0 | GRAND BOULEVARD | 0.0 | 4.849250e+07 | 28196.837157 | POLYGON ((-87.60671 41.81681, -87.60670 41.816... | ... | 1.0 | 1454.6 | 1680 | 13.2 | 28.3 | 2.7 | 41.7 | 19.4 | 22056 | 20.6 |
| **4** | 0.0 | 39 | 39 | 0.0 | 0.0 | KENWOOD | 0.0 | 2.907174e+07 | 23325.167906 | POLYGON ((-87.59215 41.81693, -87.59215 41.816... | ... | 0.4 | 610.2 | 549.1 | 0.0 | 23.1 | 2.3 | 34.2 | 10.8 | 37519 | 11.0 |

5 rows × 39 columns

### 2.1.5.1 公共健康数据的简单线性回归

在进行回归之前，需要做相关性分析，确定解释变量和响应变量是否存在相关性。返回到“公共健康数据的相关性分析”部分查看已经计算的结果，在经济条件数据和疾病数据中选取“Per Capita Income”（人均收入）和“Childhood Blood Lead Level Screening”（儿童血铅水平检查），其相关系数为-0.64，呈现线性负相关关系。在回归部分阐述过求取简单线性回归方程的方法为普通最小二乘法（Ordinary Lease Squares,OLS）或线性最小二乘，即通过最小化残差平方和，对回归系数求微分并另微分结果为0，解方程组得回归系数值，构建简单线性回归方程，或模型、估计器。如果模型预测的响应变量都接近观测值，那么模型就是拟合的，用残差平方和来衡量模型拟合性的方法称为残差平方和（RSS）代价函数。代价函数也被称为损失函数，用于定义和衡量一个模型的误差。其公式同回归部分的残差平方和： 或，式中yi​为观测值，​​和f(xi​)均为预测值（回归值），f(xi​)即为所求得的估计器。因此对残差平方和回归系数求微分的过程就是对损失函数求极小值找到模型参数值的过程，二者只是表述上的不同。

在*Mastering Machine Learning with scikit-learn*[1]中，作者给出了另一种求解简单线性回归系数的方法，计算斜率的公式为：，其中var(x)为解释变量的方差， n为训练数据的总量； cov(x,y)为协方差， xi​表示训练数据集中第i个x的值，为解释变量的均值， yi​为第i个y的值，​为响应变量的均值。求得β之后，再由公式： α=​−βx求得α截距。计算结果与使用sklearn的LinearRegression（OLS）方法保持一致。

方差（variance）用来衡量一组值的偏离程度，通常标记为σ2，即标准差的平方，或方差的平方根即为标准差。当集合中的所有数值都相等时，其方差为0。方差描述了一个随机变量离其期望值的距离，通常期望值为该组值的均值。

协方差（covariance）用来衡量两个变量的联合变化程度。方差则是协方差的一种特殊情况，即变量与自身的协方差。协方差表示两个变量总体的误差，这与只表示一个变量误差的方差不同。如果两个变量的变化趋势一致，即其中一个大于自身的期望值，另一个也大于其自身的期望值，则两个变量之间的协方差为正值；如果两个变量的变化趋势相反，即其中一个大于自身期望值，而另一个小于自身期望值，则两个变量之间的协方差为负值；当变量所有值都趋近于各自的期望值时，协方差趋近于0。两个变量统计独立时，协方差为0。但是协方差为0，不仅包括统计独立一种情况，如果两个变量间没有线性关系，二者之间的协方差为0，其线性无关但不一定是相对独立。

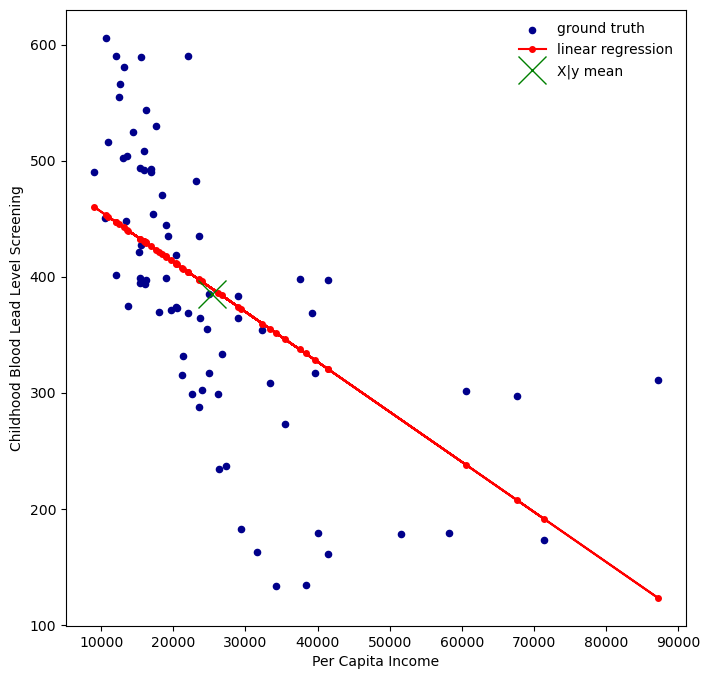
* 训练（数据）集，测试（数据）集和验证（数据）集

通常在训练模型前将数据集划分为训练数据集，测试数据集，也经常增加有验证数据集。训练数据集用于模型的训练，测试数据集依据衡量标准用来评估模型性能，测试数据集不能包含训练数据集中的数据，否则很难评估算法是否真的从训练数据集中学到了泛化能力，还是只是简单的记住了训练的样本。一个能够很好泛化的模型可以有效的对新数据进行预测。如果只是记住了训练数据集中解释变量和响应变量之间的关系，则称为过拟合。对于过拟合通常用正则化的方法加以处理。验证数据集常用来微调超参数的变量，超参数通常由人为调整配置，控制算法如何从训练数据中学习。各自部分划分没有固定的比例，通常训练集占50%∼∼75%，测试集占10%∼∼25%，余下的为验证集。

决定系数的计算结果为0.475165，其分数并没有过0.5，所训练的简单线性回归模型并不能根据解释变量很好的预测响应变量。事实上，在现实的世界中，很少用到简单的线性回归模型，数据的复杂性往往需要借助更有利的算法来解决实际问题。但是简单线性回归的逐步计算方式的阐述，可以对回归模型有个比较清晰的理解。

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
  
ax1=pubicHealth\_Statistic.plot.scatter(x='Per Capita Income',y='Childhood Blood Lead Level Screening',c='DarkBlue',figsize=(8,8),label='ground truth')  
data\_IncomeLead=pubicHealth\_Statistic[['Per Capita Income','Childhood Blood Lead Level Screening']].dropna() # sklearn求解模型需要移除数据集中的空值  
X=data\_IncomeLead['Per Capita Income'].to\_numpy().reshape(-1,1) # 将特征值数据格转换为numpy的特征向量矩阵  
y=data\_IncomeLead['Childhood Blood Lead Level Screening'].to\_numpy() # 将目标值数据格式转换为numpy格式的向量  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)  
  
from sklearn.linear\_model import LinearRegression  
# 构建与拟合模型  
LR=LinearRegression().fit(X\_train,y\_train) #  
# 模型参数  
print("\_"\*50)  
print("sklearn slop:%.6f,intercept(b):%.6f"%(LR.coef\_, LR.intercept\_))  
ax1.plot(X\_train,LR.predict(X\_train),'o-',label='linear regression',color='r',markersize=4)  
ax1.set(xlabel='Per Capita Income',ylabel='Childhood Blood Lead Level Screening')  
  
# 逐步计算回归系数  
income\_variance=np.cov(X\_train.T) # 用nu.cov()求方程及协方差，注意返回值，如果是对两个变量求协方差，返回值为各自变量的方差及两个变量的协方差的矩阵  
income\_lead\_covariance=np.cov(X\_train.T,y\_train)  
print("income\_variance=%.6f,income\_lead\_covariance=%.6f"%(income\_variance,income\_lead\_covariance[0,1]))  
beta=income\_lead\_covariance[0,1]/income\_variance  
alpha=y\_train.mean()-beta\*X\_train.T.mean()  
print("beta=%.6f,alpha=%.6f"%(beta,alpha))  
  
ax1.plot(X.T.mean(),y.mean(),'x',label='X|y mean',color='green',markersize=20)  
ax1.legend(loc='upper right', frameon=False)  
plt.show()  
  
# 简单线性回归方程-回归显著性检验（回归系数检验）  
print("决定系数，coefficient of determination，r\_squared=%.6f"%LR.score(X\_test,y\_test))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
sklearn slop:-0.004310,intercept(b):499.029224  
income\_variance=266830859.633061,income\_lead\_covariance=-1150133.371184  
beta=-0.004310,alpha=499.029224



决定系数，coefficient of determination，r\_squared=0.475165

### 2.1.5.2 公共健康数据的K-近邻模型（k-nearest neighbors algorithm，k-NN）

#### 1）k-NN

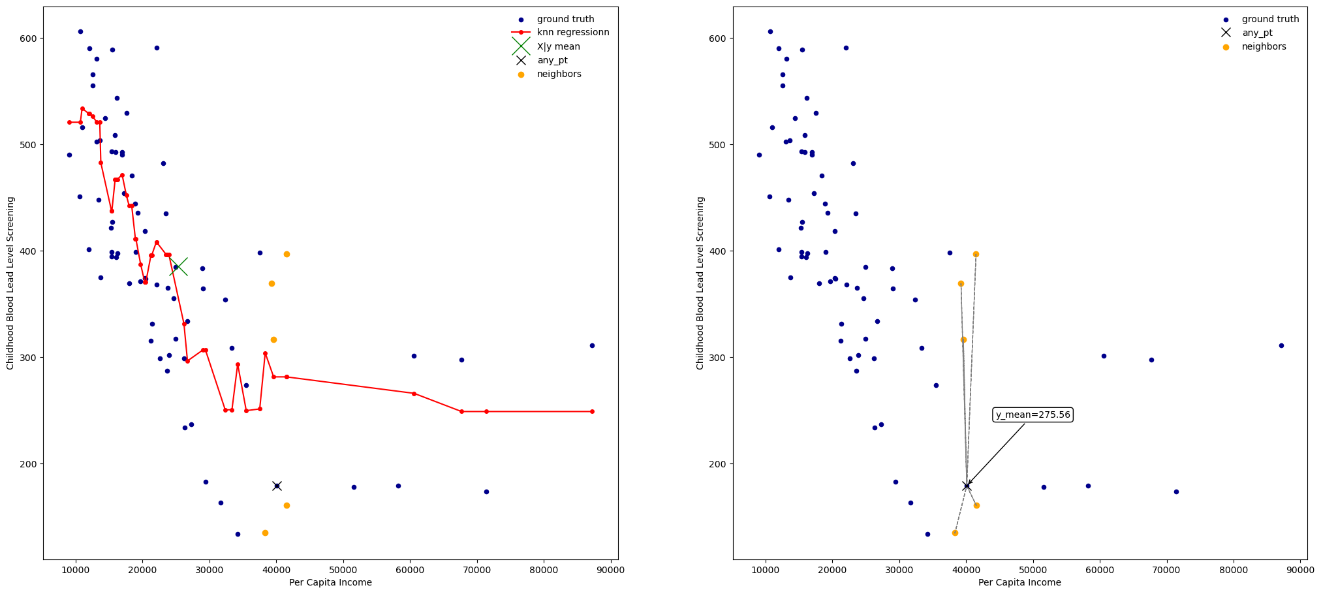
k-NN是一种用于回归任务和分类任务的简单模型。在2、3维空间中可以将解释变量与响应变量映射到可以观察的空间中，这个能够定义数据集中所有成员之间距离的特征空间就是度量空间，但是如果解释变量数量比较多，就很难映射变量到可以观察的2，3维空间中，而是更高的空间维度。所有变量的成员都可以在这个度量空间中表示。k-NN的原理就是找到邻近的样本，在分类任务中是看这些样本对应占据多数的响应变量的类别；而回归任务中，则是计算邻近样本响应变量的均值。决定系数r\_squared=0.675250。用k-NN算法求得的回归模型较之简单线性回归模型的预测能力有很大改善，表明’Childhood Blood Lead Level Screening’儿童血铅水平检查变量的方差很大比例上可以被模型解释。

k-NN中成员之间的距离（或理解为两点之间或多点之间的距离），通常使用欧式（欧几里得）距离（Euclidean Distance），对于n维空间中两个点x1​(x11​,x12​,…,x1n​)和x2​(x21​,x22​,…,x2n​)间的欧式距离公式为：，二维空间中的欧式距离公式即为直角三角形的勾股定理：。

同时，sklearn提供了权重（weight）参数，这对于一些受城市空间地理位置影响的变量分析尤其有用，而[PySAL](https://pysal.org/)②库在空间权重部分有大量可以直接应用的方法。在权重参数配置上，包含三个可选参数，分别为unifom，每个邻域内所有点的权重是相等的；distance，权重为距离的倒数，即越接近查询点的邻居点权重越高，越远的则越低；callable，用户定义权重。对于本次人均收入与儿童血铅水平检查之间的模型建立，因为并未涉及到空间地理位置的影响分析，因此参数配置为默认，即uniform模式。

# 使用skearn库训练k-NN  
from sklearn.neighbors import KNeighborsRegressor  
import math  
fig, axs=plt.subplots(1,2,figsize=(25,11))  
  
k\_neighbors=range(1,15,1)  
r\_squared\_temp=0  
for k in k\_neighbors:  
 knn=KNeighborsRegressor (n\_neighbors=k)  
 knn.fit(X\_train,y\_train)  
 if r\_squared\_temp<knn.score(X\_test,y\_test): # knn-回归显著性检验（回归系数检验）  
 r\_squared\_temp=knn.score(X\_test,y\_test)   
 k\_temp=k  
knn=KNeighborsRegressor (n\_neighbors=k\_temp).fit(X\_train,y\_train)  
print("在区间%s,最大的r\_squared=%.6f,对应的k=%d"%(k\_neighbors,knn.score(X\_test,y\_test) ,k\_temp))  
  
pubicHealth\_Statistic.plot.scatter(x='Per Capita Income',y='Childhood Blood Lead Level Screening',c='DarkBlue',label='ground truth',ax=axs[0])  
X\_train\_sort=np.sort(X\_train,axis=0)  
axs[0].plot(X\_train\_sort,knn.predict(X\_train\_sort),'o-',label='knn regressionn',color='r',markersize=4)  
  
axs[0].set(xlabel='Per Capita Income',ylabel='Childhood Blood Lead Level Screening')  
axs[0].plot(X.T.mean(),y.mean(),'x',label='X|y mean',color='green',markersize=20)  
  
# 关于K-近邻  
Xy=data\_IncomeLead.to\_numpy()  
# A - 自定义返回K-近邻点索引的函数  
def k\_neighbors\_entire(xy,k=3):   
 '''  
 function - 返回指定邻近数目的最近点坐标  
   
 Params:  
 xy - 点坐标二维数组，例如  
 array([[23714. , 364.7],  
 [21375. , 331.4],  
 [32355. , 353.7],  
 [35503. , 273.3]]  
 k - 指定邻近数目；int  
   
 return:  
 neighbors - 返回各个点索引，及各个点所有指定数目邻近点索引；list(tuple)  
 '''  
 import numpy as np  
   
 neighbors=[(s,np.sqrt(np.sum((xy-xy[s])\*\*2,axis=1)).argsort()[1:k+1]) for s in range(xy.shape[0])]  
 return neighbors  
   
neighbors=k\_neighbors\_entire(Xy,k=5)  
any\_pt\_idx=70  
any\_pt=np.take(Xy,neighbors[any\_pt\_idx][0],axis=0)  
neighbor\_pts=np.take(Xy,neighbors[any\_pt\_idx][1],axis=0)  
  
axs[0].plot(any\_pt[0],any\_pt[1],'x',label='any\_pt',color='black',markersize=10)  
axs[0].scatter(neighbor\_pts[:,0],neighbor\_pts[:,1],c='orange',label='neighbors')  
  
# B - 用 PySAL库下的pointpats求得近邻点  
from pointpats import PointPattern  
pubicHealth\_Statistic.plot.scatter(x='Per Capita Income',y='Childhood Blood Lead Level Screening',c='DarkBlue',label='ground truth',ax=axs[1])  
Xy=data\_IncomeLead.to\_numpy()  
pp=PointPattern(Xy)  
pp\_neighbor=pp.knn(5)  
  
any\_pt=np.take(Xy,any\_pt\_idx,axis=0)  
neighbor\_pts\_idx=np.take(pp\_neighbor[0],any\_pt\_idx,axis=0)  
neighbor\_pts=np.take(Xy,neighbor\_pts\_idx,axis=0)  
  
axs[1].plot(any\_pt[0],any\_pt[1],'x',label='any\_pt',color='black',markersize=10)  
axs[1].scatter(neighbor\_pts[:,0],neighbor\_pts[:,1],c='orange',label='neighbors')  
for coodi in neighbor\_pts:  
 axs[1].arrow(any\_pt[0],any\_pt[1],coodi[0]-any\_pt[0], coodi[1]-any\_pt[1], head\_width=1, head\_length=1,color="gray",linestyle="--" ,length\_includes\_head=True)  
   
from matplotlib.text import OffsetFrom  
axs[1].annotate("y\_mean=%s"%neighbor\_pts[:,1].mean(), xy=any\_pt, xycoords="data",xytext=(50000, 250), va="top", ha="center",bbox=dict(boxstyle="round", fc="w"),arrowprops=dict(arrowstyle="->"))   
  
axs[0].legend(loc='upper right', frameon=False)  
axs[1].legend(loc='upper right', frameon=False)  
plt.show()

在区间range(1, 15),最大的r\_squared=0.675250,对应的k=5



#### 2）平均绝对误差（mean absolute error, MAE）和均方误差（mean squared error，MSE）

平均绝对误差（MAE），又被称为L1范数损失（L1\_Loss），是预测结果误差绝对值的均值，一种回归损失函数，公式为：，虽然能够较好的衡量回归模型的好坏，但是绝对值的存在导致函数不光滑，虽然对于什么样的输入值，都有稳定的梯度，不会导致梯度爆炸问题，具有较为稳健型的解，但是在中心点是折点，不能求导，不方便求解。

均方误差（MSE），又称均方偏差，比起MAE来说是一种更为常用的指标，也称为L2​范数损失（L2\_Loss），是最为常用的损失函数，即为在回归部分阐述的残差平方和(SSres​)再除以样本容量，公式为：。L2​范数损失MSE，各点连续光滑，方便求导，具有较为稳定的解。但是不是特别稳健，当函数的输入值距离中心值较远时，使用梯度下降法求解时梯度很大，导致在神经网络训练过程中网络权重的大幅度更新，使得网络变得不稳定，极端情况下，权值值可能溢出，即梯度爆炸。

从下述的图形中，通过打印MAE和MSE的函数图形，更容易观察到函数的变化。回归部分对残差平方和的，求微分，并令其结果等于0，求得回归方程。扩展到sklearn机器学习部分，则是对损失函数求解，这样能够更容易理解什么是损失函数。注意下述代码建立的简单线性回归方程为：，并没有加截距，以便简化运算。

范数（norm），是具有“长度”概念的函数。在线性代数、泛函分析及相关的数学领域，是一个函数，为向量空间内的所有向量赋予非零的正长度或大小。例如，一个二维度的欧式几何空间R²就有欧式范数。在这个向量空间的元素 (xi​,yi​)，常常在笛卡尔坐标系统被画成一个从原点出发的箭号，每一个向量的欧式范数就是箭号的长度。或者理解为向量空间（即度量空间）中的向量都是有大小，度量这个大小的方式就是用范数来度量，不同的范数都可以来度量这个大小，其范数的定义为：向量的范数是一个函数，满足非负性，齐次性，三角不等式。

在阐述k-NN时，谈到度量空间，对于2、3维的向量可以画出几何图形，但是对于多维、高维，超出三维空间时，则引入范数的概念，从而可以定义任意维度两个向量的距离。常用向量的范数有：

1.L1​范数：为x向量各个元素绝对值之和，公式为：

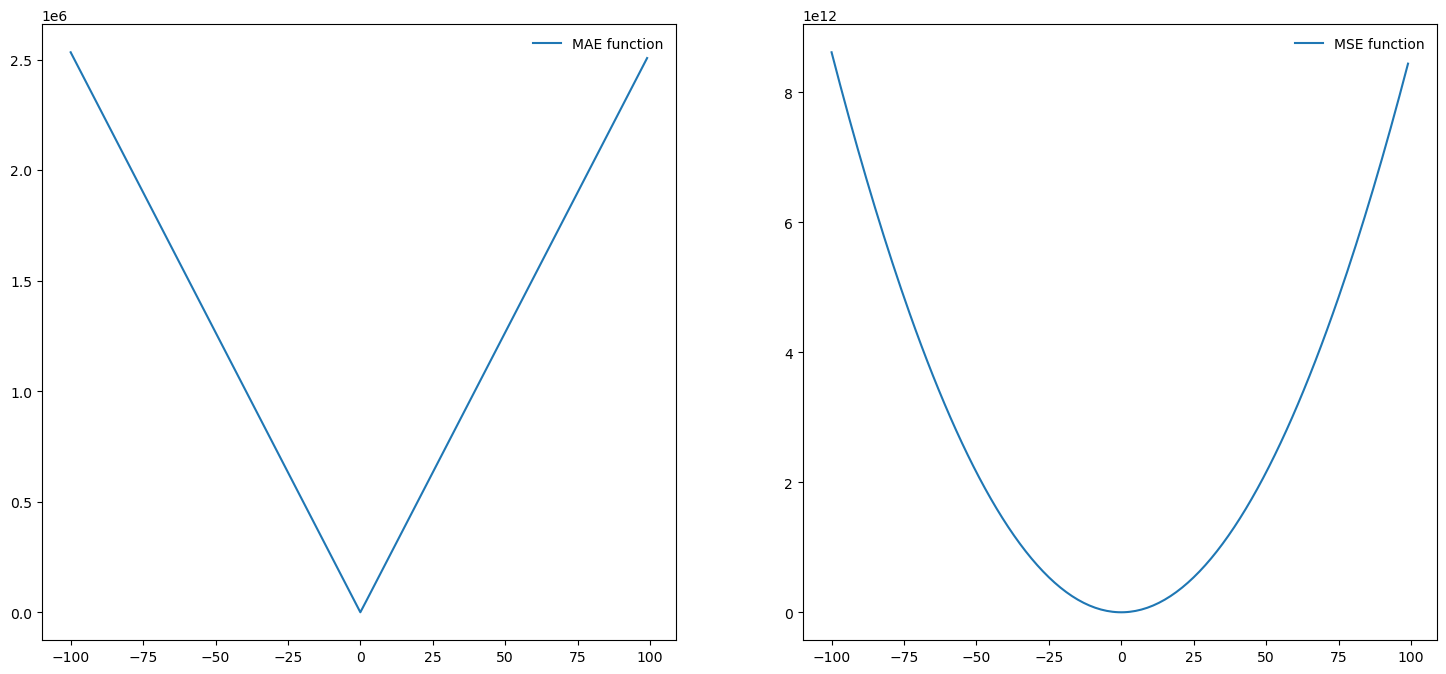
2.L2​范数：为x向量各个元素平方和的1/2次方，公式为：

3.Lp​范数：为x向量各个元素绝对值p次方和的1/p次方，公式为：

4.L∘范数：为x向量各个元素绝对值最大那个元素的绝对值。

from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error,r2\_score  
print("coefficient of determination=%s"%r2\_score(y\_test,knn.predict(X\_test)))  
print("mean absolute error=%s"%mean\_absolute\_error(y\_test,knn.predict(X\_test)))  
print("mean squared error=%s"%mean\_squared\_error(y\_test,knn.predict(X\_test)))  
  
# 打印MAE函数图形  
fig, axs=plt.subplots(1,2,figsize=(18,8))  
  
import sympy  
import pandas as pd  
a=sympy.symbols('a')  
data\_IncomeLead\_copy=data\_IncomeLead.copy(deep=True)  
data\_IncomeLead\_copy.rename(columns={"Per Capita Income":"PCI","Childhood Blood Lead Level Screening":"CBLLS"},inplace=True)  
  
data\_IncomeLead\_copy["residual"]=data\_IncomeLead\_copy.apply(lambda row:row.CBLLS-(a\*row.PCI),axis=1)  
data\_IncomeLead\_copy["abs\_residual"]=data\_IncomeLead\_copy.residual.apply(lambda row:abs(row))  
n\_s=data\_IncomeLead\_copy.shape[0]  
MAE=data\_IncomeLead\_copy.abs\_residual.sum()/n\_s  
MAE\_=sympy.lambdify([a],MAE,"numpy")  
a\_val=np.arange(-100,100,1) # 假设的a值  
  
axs[0].plot(a\_val,MAE\_(a\_val),'-',label='MAE function')  
  
# 打印MSE函数图形  
data\_IncomeLead\_copy["residual\_squared"]=data\_IncomeLead\_copy.residual.apply(lambda row:row\*\*2)  
MSE=data\_IncomeLead\_copy.residual\_squared.sum()/n\_s  
MSE\_=sympy.lambdify([a],MSE,"numpy")  
axs[1].plot(a\_val,MSE\_(a\_val),'-',label='MSE function')  
  
axs[0].legend(loc='upper right', frameon=False)  
axs[1].legend(loc='upper right', frameon=False)  
plt.show()

coefficient of determination=0.675249990123995  
mean absolute error=59.75384615384613  
mean squared error=4912.952276923076



#### 3）特征值的比例缩放(标准化)

人均收入的特征值范围在[9016, 87163]区间，单位为美元；儿童血铅水平检查特征值范围在[133.6, 605.9]区间，单位为每1,000名儿童（0~6岁）。两个特征向量单位不同，取值范围相差也很大，必然会影响模型训练，如果将其范围缩放到相同的取值范围下，学习算法会比较好的运行。特征值的缩放可以使用sklearn提供的sklearn.preprocessing.StandardScaler方法，其计算公式为：z=(x−μ)/s，其中μ为训练样本的均值， s为训练样本的标准差。

计算结果显示，通过特征值的标准化后其决定系数为0.72，大于特征值处理前的值0.68，模型预测能力得以再次提升。

from sklearn.preprocessing import StandardScaler  
SS=StandardScaler()  
X\_train\_scaled=SS.fit\_transform(X\_train)   
X\_test\_scaled=SS.fit\_transform(X\_test)   
  
k\_neighbors=range(1,15,1)  
r\_squared\_temp=0  
for k in k\_neighbors:  
 knn\_=KNeighborsRegressor (n\_neighbors=k)  
 knn\_.fit(X\_train\_scaled,y\_train)  
 if r\_squared\_temp<knn\_.score(X\_test\_scaled,y\_test): # knn-回归显著性检验（回归系数检验）  
 r\_squared\_temp=knn\_.score(X\_test\_scaled,y\_test)   
 k\_temp=k  
knn\_=KNeighborsRegressor(n\_neighbors=k\_temp).fit(X\_train\_scaled,y\_train)  
  
from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error,r2\_score  
print("coefficient of determination=%s"%r2\_score(y\_test,knn\_.predict(X\_test\_scaled)))  
print("mean absolute error=%s"%mean\_absolute\_error(y\_test,knn\_.predict(X\_test\_scaled)))  
print("mean squared error=%s"%mean\_squared\_error(y\_test,knn\_.predict(X\_test\_scaled)))  
  
from numpy.polynomial import polyutils as pu  
Per\_Capita\_Income\_domain=pu.getdomain(X.reshape(-1))  
Childhood\_Blood\_Lead\_Level\_Screening\_domain=pu.getdomain(y)  
print("Per Capita Income domain=%s"%Per\_Capita\_Income\_domain)  
print("Childhood Blood Lead Level Screening domain=%s"%Childhood\_Blood\_Lead\_Level\_Screening\_domain)

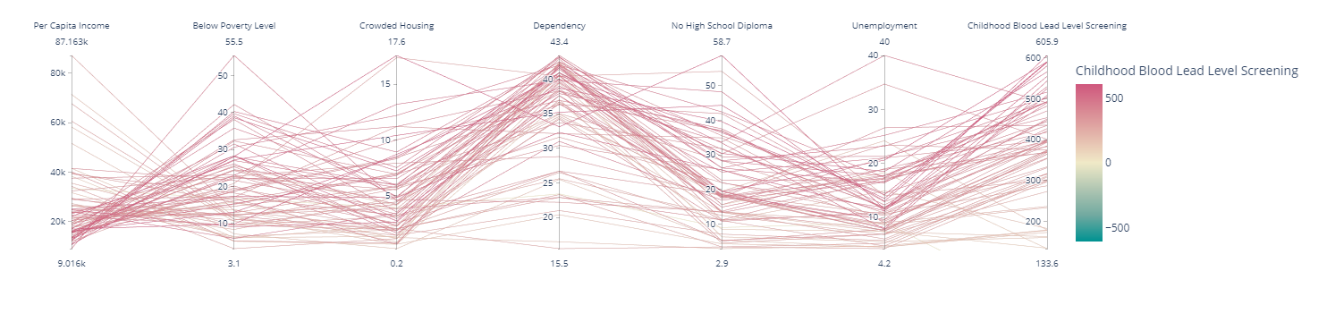
coefficient of determination=0.7150373630356484  
mean absolute error=56.32587412587413  
mean squared error=4311.032466624287  
Per Capita Income domain=[ 9016. 87163.]  
Childhood Blood Lead Level Screening domain=[133.6 605.9]

### 2.1.5.3 公共健康数据的多元回归

#### 1）多元线性回归

超过3个的多个解释变量（多维或高维数组），仍然有很多方法打印图表观察数据的关联情况，例如使用平行坐标图（parallel coordinates plot），响应变量为“Childhood Blood Lead Level Screening”，其它的为解释变量。通过折线的变化趋势能够初步判断各个解释变量之间，及与响应变量之间是正相关还是负相关，而折线的分布密度则表明各个变量的数值分布情况。

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn.linear\_model import LinearRegression  
import plotly.express as px  
  
columns=['Per Capita Income','Below Poverty Level','Crowded Housing','Dependency','No High School Diploma','Unemployment','Childhood Blood Lead Level Screening']  
data\_Income=pubicHealth\_Statistic[columns].dropna() # sklearn求解模型需要移除数据集中的空值  
  
fig=px.parallel\_coordinates(data\_Income, labels=columns,color\_continuous\_scale=px.colors.diverging.Tealrose,color\_continuous\_midpoint=2,color='Childhood Blood Lead Level Screening')  
fig.show()



由“Per Capita Income”一个特征训练回归模型的决定系数为0.48，由多个特征，此次为6个解释变量，获得的决定系数为0.76，较之单一特征有大幅度的提升，增加的解释变量提升了模型的性能。同时比较k-NN模型，当k=4时，6个特征作为解释变量，其决定系数为0.82，而一个特征时为0.72，模型的性能也得以大幅度提升，表明增加的解释变量对模型的预测是有贡献的。

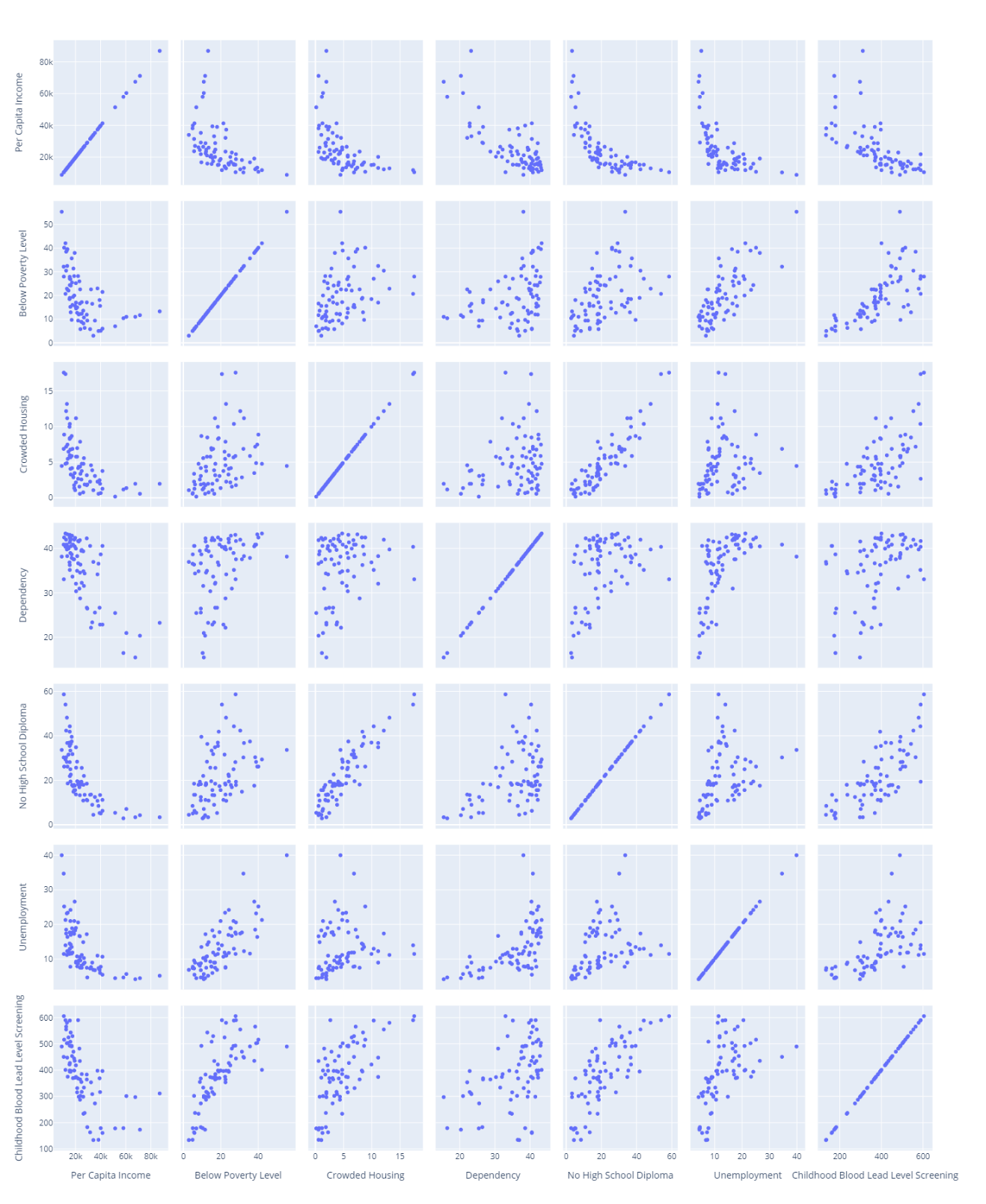
X=data\_Income[['Per Capita Income','Below Poverty Level','Crowded Housing','Dependency','No High School Diploma','Unemployment']].to\_numpy() # 将特征值数据格转换为numpy的特征向量矩阵  
y=data\_Income['Childhood Blood Lead Level Screening'].to\_numpy() # 将目标值数据格式转换为numpy格式的向量  
  
SS=StandardScaler()  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)  
X\_train\_scaled=SS.fit\_transform(X\_train)   
X\_test\_scaled=SS.fit\_transform(X\_test)  
  
LR\_m=LinearRegression()  
LR\_m.fit(X\_train\_scaled, y\_train)  
print("Linear Regression - Accuracy on test data: {:.2f}".format(LR\_m.score(X\_test\_scaled, y\_test)))  
y\_pred=LR\_m.predict(X\_train\_scaled)  
  
# 使用k-NN  
from sklearn.neighbors import KNeighborsRegressor  
k\_neighbors=range(1,15,1)  
r\_squared\_temp=0  
for k in k\_neighbors:  
 knn=KNeighborsRegressor (n\_neighbors=k)  
 knn.fit(X\_train\_scaled, y\_train)  
 if r\_squared\_temp<knn.score(X\_test\_scaled,y\_test): # knn-回归显著性检验（回归系数检验）  
 r\_squared\_temp=knn.score(X\_test\_scaled,y\_test)   
 k\_temp=k  
knn=KNeighborsRegressor (n\_neighbors=k\_temp).fit(X\_train\_scaled,y\_train)  
print("k-NN 在区间%s,最大的r\_squared=%.6f,对应的k=%d"%(k\_neighbors,knn.score(X\_test\_scaled,y\_test) ,k\_temp))

Linear Regression - Accuracy on test data: 0.76  
k-NN 在区间range(1, 15),最大的r\_squared=0.815978,对应的k=4

#### 2）多项式回归

真实的世界，很多因果关系并不是线性的，对于非线性的解释变量和响应变量之间的关系建模可以有多种途径。这里用多项式回归的方法，该方法在微分部分用于拟合模型曲线，得到的结果较之线性模型有很大提升。首先打印响应变量“Childhood Blood Lead Level Screening”与其它所有经济条件解释变量的散点图，观察发现与“Per Capita Income”特征趋于线性，而与其它的特征似乎呈现一定的弧度。

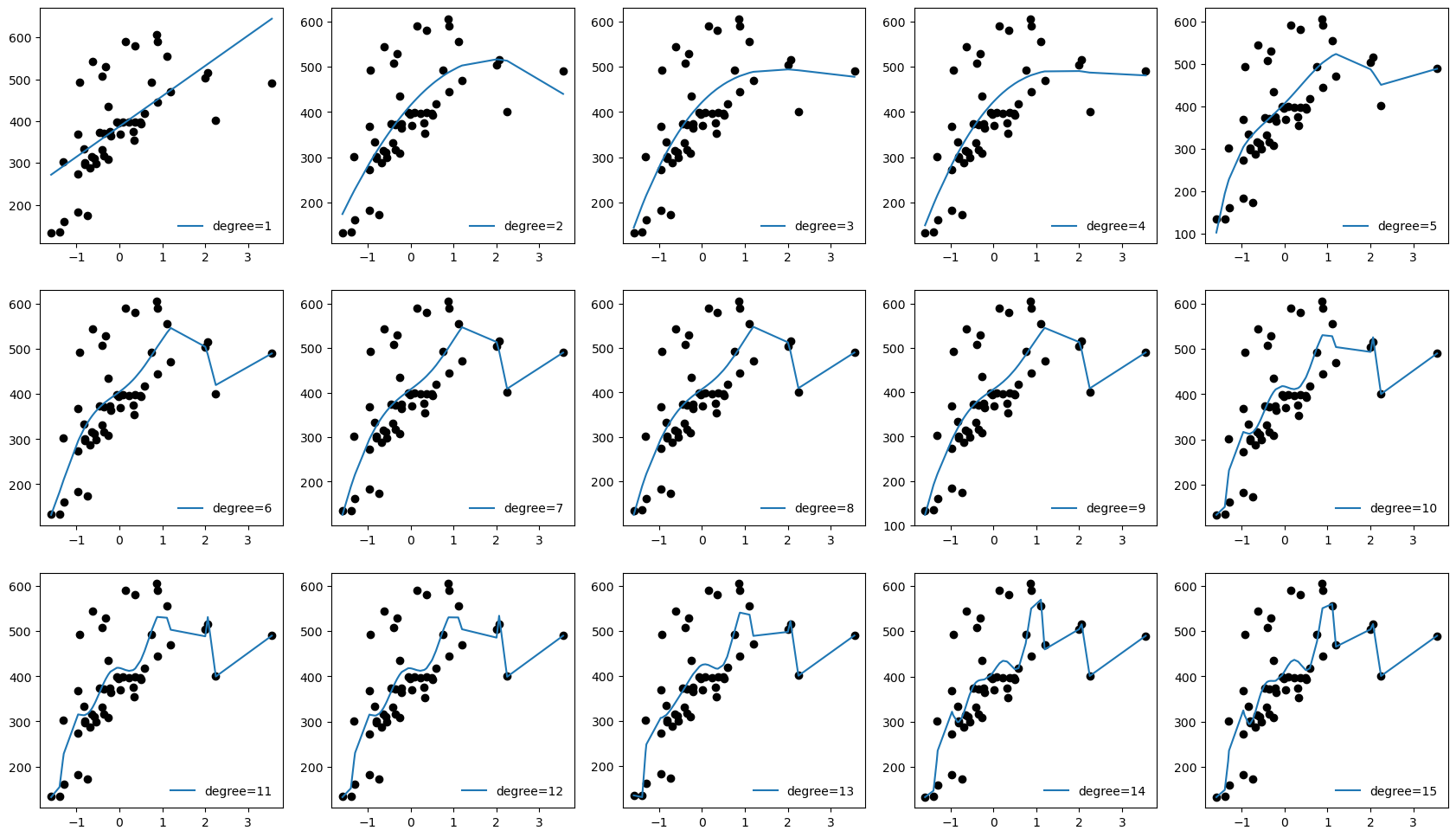
import plotly.express as px  
fig=px.scatter\_matrix(data\_Income)  
  
fig.update\_layout(  
 autosize=True,  
 width=1800,  
 height=1800,  
 )  
fig.show()



为了方便观察拟合曲线，只选择”Below Poverty Level”一个解释变量，响应变量依旧为”Childhood Blood Lead Level Screening”。定义PolynomialFeatures\_regularization()函数，用于多项式回归degree次数选择对回归结果影响的比较和讨论。包含对不同正则化方法的调用，由参数regularization配置，包括“linear”、“Ridge”和“LASSO”三种方式，分别代表不采用正则化，采取L2​正则化（又称岭回归，Ridge Regression）和L1​正则化（又称LASSO）。本小节只对不加正则化的多项式回归方法进行讨论，即配置regularization='linear'，下一节则讨论两种正则化方法对回归结果的影响。PolynomialFeatures方法的传入参数degree配置多项式特征的次数，默认为2。同时循环不同的degree值，绘制对应的拟合曲线，并通过比较决定系数的值获得值最高时的degree值。计算结果为当degree=4时，决定系数值0.81为最高。

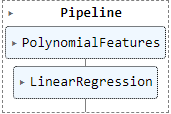
X=data\_Income['Below Poverty Level'].to\_numpy().reshape(-1,1) # 将特征值数据格转换为numpy的特征向量矩阵  
y=data\_Income['Childhood Blood Lead Level Screening'].to\_numpy() # 将目标值数据格式转换为numpy格式的向量  
  
def PolynomialFeatures\_regularization(X,y,regularization='linear'):  
 '''  
 function - 多项式回归degree次数选择，及正则化  
   
 Params:  
 X - 解释变量；array  
 y - 响应变量；array  
 regularization - 正则化方法， 为'linear'时，不进行正则化，正则化方法为'Ridge'和'LASSO'；string  
   
 Returns:  
 reg - model  
 '''   
 from sklearn.preprocessing import PolynomialFeatures  
 from sklearn.linear\_model import LinearRegression  
 from sklearn.pipeline import Pipeline  
 from sklearn.preprocessing import StandardScaler   
 from sklearn.linear\_model import Ridge  
 from sklearn.linear\_model import Lasso  
   
 X\_train, X\_test, y\_train, y\_test=train\_test\_split(X, y, test\_size=0.33, random\_state=42)  
 SS=StandardScaler()  
 X\_train\_scaled=SS.fit\_transform(X\_train)   
 X\_test\_scaled=SS.fit\_transform(X\_test)  
  
 degrees=np.arange(1,16,1)  
 fig\_row=3  
 fig\_col=degrees.shape[0]//fig\_row  
 fig, axs=plt.subplots(fig\_row,fig\_col,figsize=(21,12))  
 r\_squared\_temp=0  
 p=[(r,c) for r in range(fig\_row) for c in range(fig\_col)]  
 i=0  
 for d in degrees:  
 if regularization=='linear':  
 model=Pipeline([('poly', PolynomialFeatures(degree=d)),  
 ('regular', LinearRegression(fit\_intercept=False))])  
 elif regularization=='Ridge':  
 model=Pipeline([('poly', PolynomialFeatures(degree=d)),  
 ('regular', Ridge())])   
 elif regularization=='LASSO':  
 model=Pipeline([('poly', PolynomialFeatures(degree=d)),  
 ('regular', Lasso())])   
   
 reg=model.fit(X\_train\_scaled,y\_train)  
 x\_=X\_train\_scaled.reshape(-1)  
 print("训练数据集的-r\_squared=%.6f,测试数据集的-r\_squared=%.6f,对应的degree=%d"%(reg.score(X\_train\_scaled,y\_train),reg.score(X\_test\_scaled,y\_test) ,d))   
 print("系数:",reg['regular'].coef\_)   
 print("\_"\*50)  
   
 X\_train\_scaled\_sort=np.sort(X\_train\_scaled,axis=0)  
  
 axs[p[i][0]][p[i][1]].scatter(X\_train\_scaled.reshape(-1),y\_train,c='black')  
 axs[p[i][0]][p[i][1]].plot(X\_train\_scaled\_sort.reshape(-1),reg.predict(X\_train\_scaled\_sort),label='degree=%s'%d)  
 axs[p[i][0]][p[i][1]].legend(loc='lower right', frameon=False)  
  
 if r\_squared\_temp<reg.score(X\_test\_scaled,y\_test): # knn-回归显著性检验（回归系数检验）  
 r\_squared\_temp=reg.score(X\_test\_scaled,y\_test)   
 d\_temp=d   
 i+=1   
  
 plt.show()   
 model=Pipeline([('poly', PolynomialFeatures(degree=d\_temp)),  
 ('linear', LinearRegression(fit\_intercept=False))])  
 reg=model.fit(X\_train\_scaled,y\_train)   
 print("\_"\*50)  
 print("在区间%s,最大的r\_squared=%.6f,对应的degree=%d"%(degrees,reg.score(X\_test\_scaled,y\_test) ,d\_temp))   
   
 return reg  
  
reg=PolynomialFeatures\_regularization(X,y,regularization='linear')

训练数据集的-r\_squared=0.383777,测试数据集的-r\_squared=0.649537,对应的degree=1  
系数: [387.116 72.44793532]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.525068,测试数据集的-r\_squared=0.799680,对应的degree=2  
系数: [415.15064819 106.59487426 -28.03464819]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.533993,测试数据集的-r\_squared=0.812620,对应的degree=3  
系数: [421.5556279 97.40720189 -40.42263154 4.91204797]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.534236,测试数据集的-r\_squared=0.812928,对应的degree=4  
系数: [422.04661766 100.89409119 -41.34031714 2.74722153 0.61986186]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.552199,测试数据集的-r\_squared=0.782880,对应的degree=5  
系数: [406.48814993 105.12024109 21.14787779 -4.00537944 -22.95458517  
 5.78979431]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.560669,测试数据集的-r\_squared=0.736790,对应的degree=6  
系数: [404.61456156 70.62356274 25.41030875 59.54880252 -30.36086237  
 -13.31892642 4.58301824]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.561443,测试数据集的-r\_squared=0.721949,对应的degree=7  
系数: [407.30609959 71.55476046 5.22112238 57.88741266 -8.34375345  
 -15.14286132 -1.01076346 1.27306864]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.561457,测试数据集的-r\_squared=0.720343,对应的degree=8  
系数: [ 4.07432749e+02 6.99138411e+01 3.76518801e+00 6.27705806e+01  
 -6.78783742e+00 -1.87285260e+01 -1.02408692e+00 2.05015447e+00  
 -1.55136817e-01]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.561509,测试数据集的-r\_squared=0.729657,对应的degree=9  
系数: [406.45186965 68.10329352 15.11566172 70.56903136 -27.35899397  
 -24.53485275 10.65409882 2.10527231 -2.31237115 0.40977271]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.584794,测试数据集的-r\_squared=0.515215,对应的degree=10  
系数: [ 416.9565451 -22.79512573 -161.63613274 587.51387394 385.40821889  
 -779.51922595 -228.82928612 393.70946989 3.93889364 -67.9693672  
 12.58496909]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.584989,测试数据集的-r\_squared=0.417839,对应的degree=11  
系数: [ 418.48831932 -17.40245655 -189.24329769 547.4857293 475.21308132  
 -712.55205506 -333.3400623 361.029845 52.98674511 -67.1027363  
 4.49118821 1.46137896]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.585052,测试数据集的-r\_squared=0.382826,对应的degree=12  
系数: [ 417.72971214 -14.7899818 -173.36519353 525.70034606 418.25351445  
 -658.55342186 -265.56473889 304.02297797 24.32398422 -40.78607186  
 6.51356956 -2.88642496 0.71844643]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.590331,测试数据集的-r\_squared=-0.114724,对应的degree=13  
系数: [ 424.34973116 42.62084484 -308.78664682 -18.49939374  
 970.0606262 653.99458524 -1197.12058714 -902.98494206  
 805.19427479 392.78276667 -310.98336871 -35.52661013  
 49.33246866 -7.44339216]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.603513,测试数据集的-r\_squared=-7.930648,对应的degree=14  
系数: [ 409.97673426 135.20495805 146.41410019 -1025.582428  
 -1545.59988052 3716.12408892 3485.70385919 -5011.48546075  
 -2794.2631344 3187.49879357 790.63782019 -988.43720627  
 -6.87130116 120.06999936 -19.88476233]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.604060,测试数据集的-r\_squared=-3.340680,对应的degree=15  
系数: [ 4.11556610e+02 1.61553839e+02 8.85250923e+01 -1.38796631e+03  
 -1.12016163e+03 5.09282046e+03 2.32199849e+03 -7.12596446e+03  
 -1.28518030e+03 4.64340360e+03 -2.16517056e+02 -1.40585554e+03  
 3.29106662e+02 1.41310366e+02 -6.37668285e+01 6.73200011e+00]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
在区间[ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15],最大的r\_squared=0.812928,对应的degree=4

reg



多个特征作为解释变量输入时，多项式回归决定系数最高为0.76，degree为1，即为线性回归模型，说明对于该数据集而言，当下述6个变量作为解释变量时，线性回归模型的预测精度要高于多项式回归。

from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import PolynomialFeatures  
  
X\_=data\_Income[['Per Capita Income','Below Poverty Level','Crowded Housing','Dependency','No High School Diploma','Unemployment',]].to\_numpy() # 将特征值数据格转换为numpy的特征向量矩阵  
y\_=data\_Income['Childhood Blood Lead Level Screening'].to\_numpy() # 将目标值数据格式转换为numpy格式的向量  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_, y\_, test\_size=0.33, random\_state=42)  
SS=StandardScaler()  
X\_train\_scaled=SS.fit\_transform(X\_train)   
X\_test\_scaled=SS.fit\_transform(X\_test)  
  
degrees=np.arange(1,16,1)  
r\_squared\_temp=0  
for d in degrees:  
 model=Pipeline([('poly', PolynomialFeatures(degree=d)),  
 ('linear', LinearRegression(fit\_intercept=False))])  
 reg=model.fit(X\_train\_scaled,y\_train)  
 x\_=X\_train\_scaled.reshape(-1)  
   
 if r\_squared\_temp<reg.score(X\_test\_scaled,y\_test): # knn-回归显著性检验（回归系数检验）  
 r\_squared\_temp=reg.score(X\_test\_scaled,y\_test)   
 d\_temp=d   
   
model=Pipeline([('poly', PolynomialFeatures(degree=d\_temp)),  
 ('linear', LinearRegression(fit\_intercept=False))])  
reg=model.fit(X\_train\_scaled,y\_train)   
print("在区间%s,最大的r\_squared=%.6f,对应的degree=%d"%(degrees,reg.score(X\_test\_scaled,y\_test) ,d\_temp))

在区间[ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15],最大的r\_squared=0.764838,对应的degree=1

#### 3）正则化

多项式回归中，计算决定系数时，在训练数据集和测试数据集上同时进行，由其结果观察到，当degree，即变量次数（阶数）增加时，训练数据集上的决定系数是增加的，但是测试数据集上所获得的决定系数并不与训练数据集的变化保持一致。这个问题称为过拟合。因为求得的单个参数数值可能非常大，当模型面对全新数据时就会产生很大波动，是模型含有巨大方差误差的问题。这往往是由于样本的特征很多，样本的容量却很少，模型就容易陷入过拟合。本次实验的特征数为6，样本容量为76，样本容量相对较少。解决过拟合的方法，一种是减少特征数量，或增加样本容量；另一种是正则化。

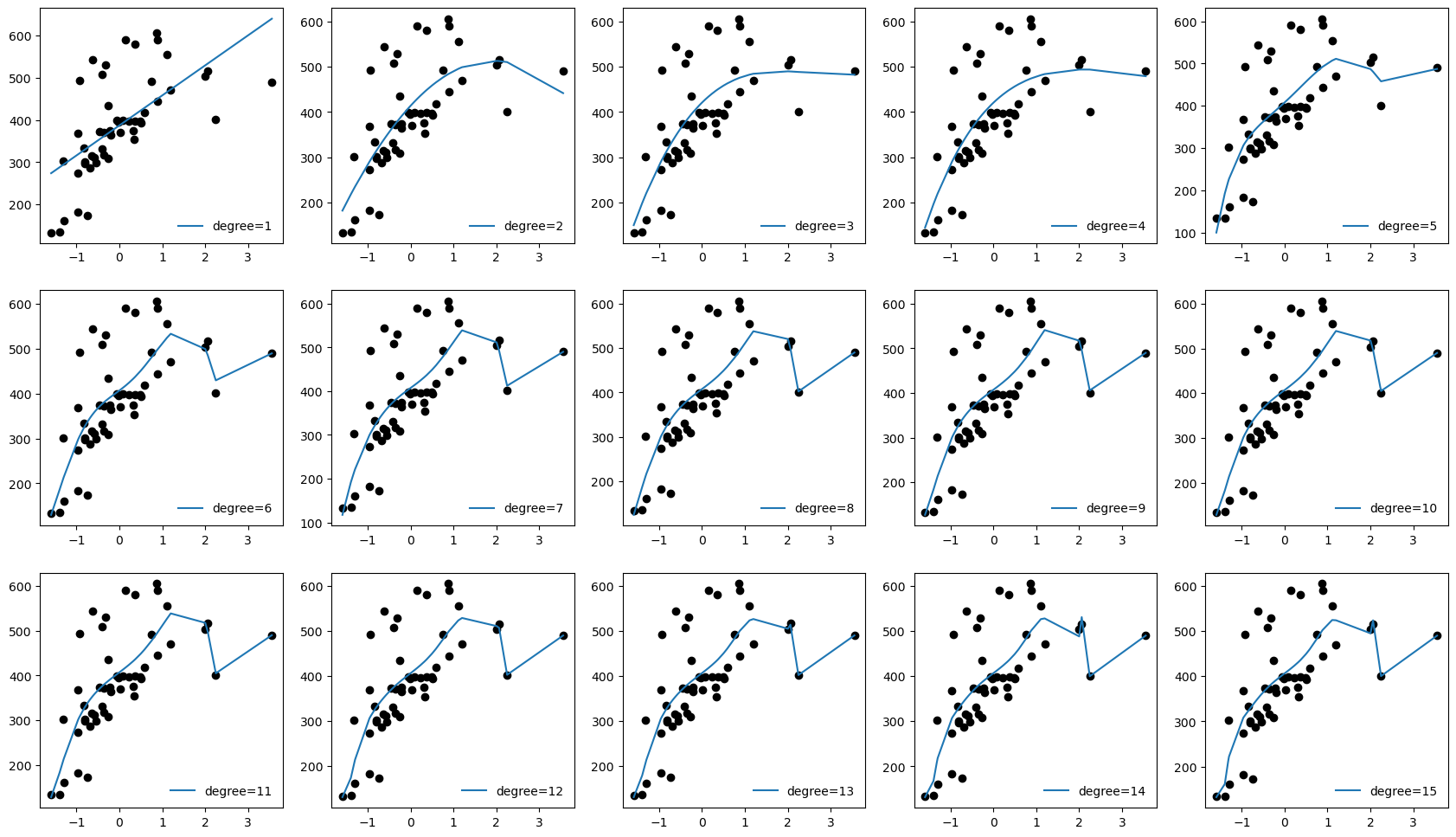
正则化是一个能用于防止过拟合的技巧集合，sklearn提供了Ridge回归方法，即L2 L1正则化的方法。对于多元线性回归模型：，其一个样本X(i)的预测值为：  
， 模型最终要求解参数A=(a0​,a1​,…,an​)T，使得均方误差MSE尽可能小。但是通过上述实验，从所获取的系数中观察到，有些系数比较大，对于测试数据集预测值可能将会有很大波动，因此为了模型的泛化能力，对参数a加以限制，岭回归通过增加系数的L2​范数来修改RSS损失函数，其公式为：。注意，前文的手动实现里， a0​是被放到最后一项，且值为1，即a0​=1，这样是为了和y=ax+b的形式呼应。

通过岭回归正则化的方式，打印计算结果及图形，可以发现训练数据集上的决定系数已经不具有明显随degree参数增加而增加的趋势，相对比较稳定。从打印的图形中也可以看到拟合曲线较之未作正则化之前平滑。最高的决定系数基本没有变化，为0.812620，但是degree的值由4变为3。

同时可以修正Ridge的参数alpha正则化的强度，来进一步优化。值越大，正则化越强。当alpha参数足够大，损失函数中起作用的几乎为正则项，曲线会成为与轴平行的直线。其默认值为1.0。

reg=PolynomialFeatures\_regularization(X,y,regularization='Ridge')

训练数据集的-r\_squared=0.383629,测试数据集的-r\_squared=0.643434,对应的degree=1  
系数: [ 0. 71.02738756]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.524482,测试数据集的-r\_squared=0.795372,对应的degree=2  
系数: [ 0. 103.02552946 -26.7958948 ]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.533428,测试数据集的-r\_squared=0.808643,对应的degree=3  
系数: [ 0. 93.50225814 -39.86167095 5.18553358]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.532731,测试数据集的-r\_squared=0.808167,对应的degree=4  
系数: [ 0. 90.14428324 -38.91589082 7.34685881 -0.62571603]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.550438,测试数据集的-r\_squared=0.788672,对应的degree=5  
系数: [ 0. 93.84764775 12.87360517 1.64678343 -20.79054041  
 4.98427692]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.559629,测试数据集的-r\_squared=0.760984,对应的degree=6  
系数: [ 0. 71.60728566 16.54120019 49.61630718 -26.29291188  
 -10.22653493 3.66477484]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.560749,测试数据集的-r\_squared=0.733622,对应的degree=7  
系数: [ 0. 71.44945769 1.37430615 49.49306876 -3.83986106  
 -12.38314706 -2.68896087 1.48586176]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.560380,测试数据集的-r\_squared=0.727878,对应的degree=8  
系数: [ 0. 72.51848056 2.84961265 38.43406208 -7.03936925 -0.11063406  
 -2.99758766 -1.70728792 0.68818115]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.560436,测试数据集的-r\_squared=0.722722,对应的degree=9  
系数: [ 0. 71.74760112 2.42062079 37.28514741 -0.1208548 2.81988314  
 -9.26801727 -1.90551868 2.18622176 -0.29603236]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.560700,测试数据集的-r\_squared=0.729012,对应的degree=10  
系数: [ 0. 72.63902932 3.38850893 37.02141463 0.43466143  
 -0.72725328 -10.83295468 1.07563872 2.30663806 -0.96127219  
 0.12701471]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.560759,测试数据集的-r\_squared=0.731137,对应的degree=11  
系数: [ 0.00000000e+00 7.25863298e+01 3.70857407e+00 3.73108505e+01  
 3.23411612e-01 -5.67036199e-01 -1.16834150e+01 7.06644641e-01  
 2.99160539e+00 -9.36936724e-01 -1.92812651e-02 2.77122144e-02]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.565228,测试数据集的-r\_squared=0.775583,对应的degree=12  
系数: [ 0. 74.22746918 3.47231772 44.25823034 8.24104255  
 -3.38598315 -6.69163606 -20.68884799 -6.92544684 16.24905022  
 0.66709704 -3.60566017 0.6825645 ]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.565621,测试数据集的-r\_squared=0.775117,对应的degree=13  
系数: [ 0. 72.93631125 5.2717895 44.61305616 10.57070042  
 0.72257551 -7.76863723 -17.35066485 -16.14740348 10.97868941  
 8.21624149 -3.13373197 -0.92815529 0.29528242]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.567804,测试数据集的-r\_squared=0.718527,对应的degree=14  
系数: [ 0. 72.48980669 -0.38993595 46.38046189 9.65430321  
 3.8019007 1.84006909 -17.13548822 -4.30892405 -5.65014533  
 -5.53173847 10.67869354 0.72563396 -2.73051094 0.52193469]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.569994,测试数据集的-r\_squared=0.725188,对应的degree=15  
系数: [ 0. 72.49854347 -0.11330674 50.1999039 10.14026572  
 5.68532456 1.07771445 -22.12227329 -5.86763851 -14.14479811  
 1.71433656 18.83966612 -5.26612263 -3.85954692 1.86774214  
 -0.21511299]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

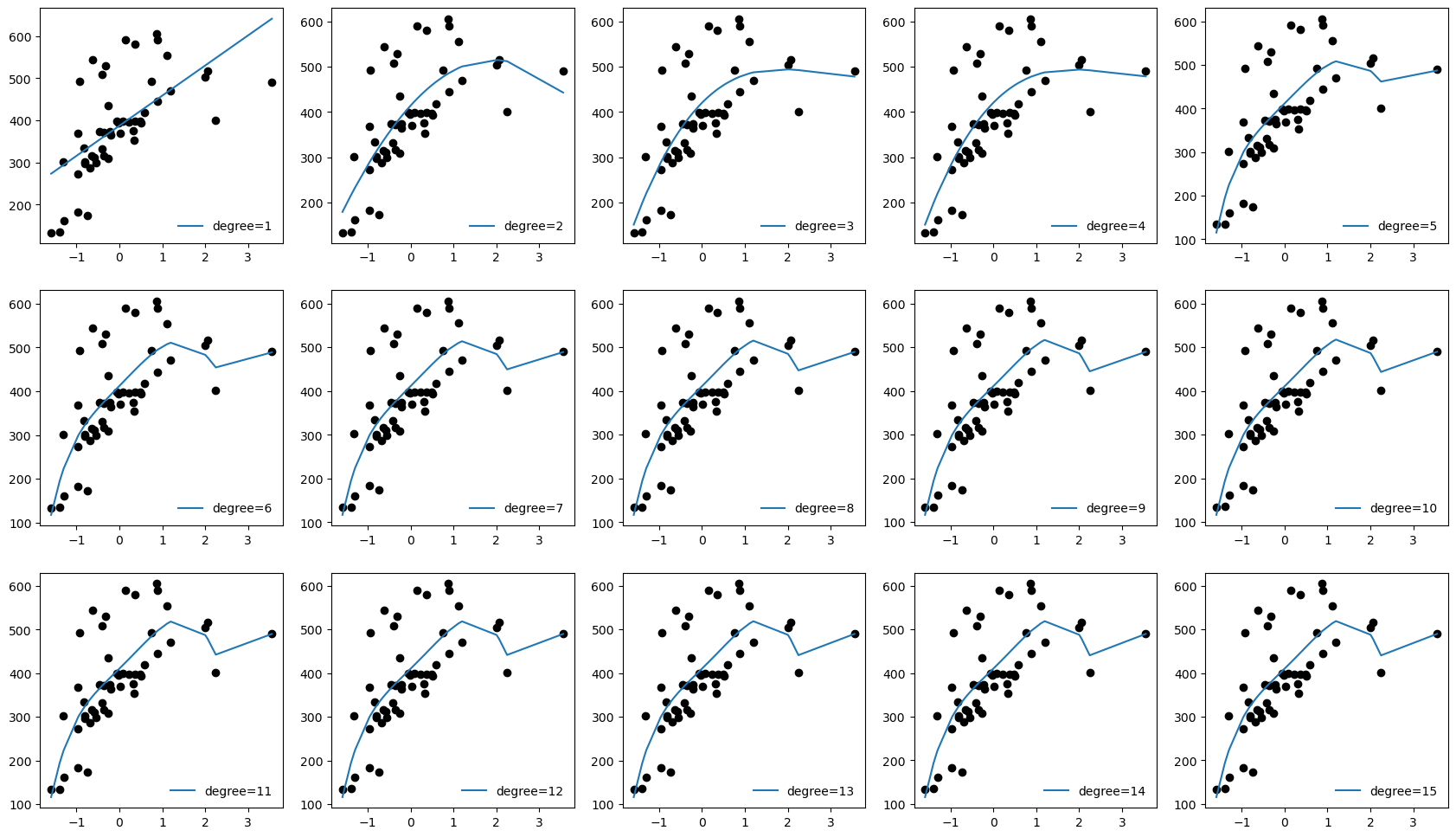


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
在区间[ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15],最大的r\_squared=0.812620,对应的degree=3

LASSO（Least Absolute Shrinkage and Selection Operator Regression）算法则通过对损失函数增加L1​范数来惩罚系数，公式为：，LASSO的特性使得部分a变为0，可以作为特征选择用，系数为0的特征说明该解释变量对模型精度的提升几乎不起作用。与Ridge在计算速度比较上，对于特征比较多的数据集，可以考虑用LASSO，否者使用Ridge，因为Ridge相对准确些。

from warnings import filterwarnings  
filterwarnings('ignore')  
reg\_lasso=PolynomialFeatures\_regularization(X,y,regularization='LASSO')

训练数据集的-r\_squared=0.383704,测试数据集的-r\_squared=0.645268,对应的degree=1  
系数: [ 0. 71.44793532]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.524848,测试数据集的-r\_squared=0.797057,对应的degree=2  
系数: [ 0. 104.49105524 -27.13097434]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.533732,测试数据集的-r\_squared=0.809695,对应的degree=3  
系数: [ 0. 95.92468724 -38.68189204 4.58082344]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.533767,测试数据集的-r\_squared=0.809769,对应的degree=4  
系数: [ 0.00000000e+00 9.60447721e+01 -3.88303966e+01 4.49967222e+00  
 3.38634247e-02]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.549919,测试数据集的-r\_squared=0.797657,对应的degree=5  
系数: [ 0. 99.25564274 0. -0. -15.43948323  
 3.85516722]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.550943,测试数据集的-r\_squared=0.798039,对应的degree=6  
系数: [ 0. 100.89815487 0.17286352 0.93205514 -15.80215342  
 3.16762738 0.19793767]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.552514,测试数据集的-r\_squared=0.794852,对应的degree=7  
系数: [ 0.00000000e+00 9.92102981e+01 1.88055289e+00 4.25201475e+00  
 -1.56479603e+01 2.14764569e+00 3.47881123e-02 1.00395677e-01]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.553184,测试数据集的-r\_squared=0.793252,对应的degree=8  
系数: [ 0.00000000e+00 9.86819406e+01 3.00767055e+00 5.40848852e+00  
 -1.58746106e+01 1.91477169e+00 1.03404514e-02 8.86030544e-02  
 9.36082835e-03]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.553865,测试数据集的-r\_squared=0.791512,对应的degree=9  
系数: [ 0.00000000e+00 9.78869938e+01 3.81158047e+00 6.77240719e+00  
 -1.59105045e+01 1.64244779e+00 -1.60745564e-02 7.80528099e-02  
 8.09780189e-03 2.79641779e-03]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.554258,测试数据集的-r\_squared=0.790384,对应的degree=10  
系数: [ 0.00000000e+00 9.74913733e+01 4.39934176e+00 7.47532494e+00  
 -1.59828872e+01 1.51928877e+00 -3.08243010e-02 7.37043881e-02  
 7.49203259e-03 2.66873015e-03 4.10414798e-04]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.554556,测试数据集的-r\_squared=0.789517,对应的degree=11  
系数: [ 0.00000000e+00 9.71471289e+01 4.76673465e+00 8.04335499e+00  
 -1.60031756e+01 1.41930136e+00 -4.17655820e-02 7.02748057e-02  
 7.02488195e-03 2.56631151e-03 3.94086203e-04 9.12164731e-05]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.554746,测试数据集的-r\_squared=0.788932,对应的degree=12  
系数: [ 0.00000000e+00 9.69412244e+01 5.03208202e+00 8.38856640e+00  
 -1.60288096e+01 1.36176783e+00 -4.83881208e-02 6.82821981e-02  
 6.74644474e-03 2.50588500e-03 3.84360223e-04 8.93139495e-05  
 1.58097622e-05]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.554878,测试数据集的-r\_squared=0.788525,对应的degree=13  
系数: [ 0.00000000e+00 9.67902373e+01 5.20020621e+00 8.63430433e+00  
 -1.60401576e+01 1.32068198e+00 -5.29313538e-02 6.68699237e-02  
 6.55130810e-03 2.46287024e-03 3.77472693e-04 8.79566428e-05  
 1.55793847e-05 3.14593156e-06]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.554965,测试数据集的-r\_squared=0.788249,对应的degree=14  
系数: [ 0.00000000e+00 9.66938208e+01 5.31829404e+00 8.79257337e+00  
 -1.60505615e+01 1.29480894e+00 -5.58620279e-02 6.59872420e-02  
 6.42723375e-03 2.43555558e-03 3.73089611e-04 8.70931670e-05  
 1.54326409e-05 3.11843117e-06 5.72801426e-07]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
训练数据集的-r\_squared=0.555024,测试数据集的-r\_squared=0.788065,对应的degree=15  
系数: [ 0.00000000e+00 9.66281005e+01 5.39448929e+00 8.89912644e+00  
 -1.60563919e+01 1.27744308e+00 -5.78371064e-02 6.53875403e-02  
 6.34304681e-03 2.41707918e-03 3.70114582e-04 8.65078199e-05  
 1.53329695e-05 3.09976035e-06 5.69537369e-07 1.08937732e-07]  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
在区间[ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15],最大的r\_squared=0.812928,对应的degree=4

### 2.1.5.4 梯度下降法（Gradient Descent）

在回归部分使用矩阵计算的方法求解模型参数值（即回归系数），其中矩阵求逆的计算较复杂，同时有些情况则无法求逆，因此引入另一种估计模型参数最优值的方法，梯度下降法。当下对于梯度下降法的解释文献繁多，主要包括通过形象的图式进行说明，使用公式推导过程等。仅有图式的说明只能大概的理解方法的过程，却对真正计算的细节无从理解；对于纯粹的公式推导，没有实例，只能空凭象形，无法落实。Udacity在线课程有一篇文章[Gradient Descent - Problem of Hiking Down a Mountain](https://jums.club/pdf/Gradient_Descent.pdf)[2]对梯度下降法的解释包括了图示、推导及实例，解释的透彻明白，因此以该文章为主导，来一步步的理解梯度下降法，这对于机器学习及深度学习有很大的帮助。

对于梯度下降法最为通用的描述是下到山谷处，找到最低点，即损失函数的最小值；在下山的过程中每一步试图找到该步通往下山路最陡路径，找到给定点的梯度，梯度的方向就是函数变化最快的方向，朝梯度反向，就能让函数值下降最快。然后不断反复这个过程，最终到达山谷。快速准确的达到山谷，需要衡量每一步在寻找最陡方向时测量的距离，如果步子过大可能错过最终山谷点，如果步子过小则增加计算的时长，并可能陷入局部最低点，因此每一步的跨度与当前地形的陡峭程度成比例，如果很陡就迈大步，如不很平缓就走小步。梯度下降法是估计函数局部最小值的优化算法。

#### 1）梯度（与微分和导数）

函数图像中，对切线斜率的理解就是导数，导数是函数图像在某点处的斜率，即纵坐标增量（△y）和横坐标增量(△x)，在△x↦0时的比值，其一般定义为，设有定义域和取值都在实数域中的函数y=f(x)，若f(x)在点x0​的某个邻域内有定义，则当自变量x在x0​处取得增量△x（点x0​+△x仍在该邻域内）时，相应的y取得增量△y=f(x0​+△x)−f(x0​)，如果△x↦0时，△y与△x之比的极限存在，则称函数y=f(x)在点x0​处可导，并称这个极限为函数y=f(x)在点x0​处的导数，记为f′(x0​)，即： ​，也可记作等。

微分是对函数的局部变化率的一种线性描述。可以近似的描述当函数自变量的取值足够小的改变时，函数的值是怎样变化的，其定义为，设函数y=f(x)在某个了邻域内有定义，对于邻域内一点x0​，当x0​变动到附近的x0​+△x（也在邻域内）时，如果函数的增量△y=f(x0​+△x)−f(x0​)可表示为△y=A△x+o(△x)，其中A是不依赖于△x的常数， o(△x)是比△x高阶的无穷小，那么称函数f(x)在点x0​是可微的，且A△x称作函数在点x0​相应于自变量增量△x的微分，记作dy，即dy=A△x， dy是△y的线性主部。通常把自变量x的增量△x称为自变量的微分，记作dx，即dx=△x。

微分和导数是两个不同的概念。但是对于一元函数来说，可微与可导是完全等价的。可微的函数，其微分等于导数乘以自变量的微分dx，即函数的微分与自变量的微分之上商等于该函数的导数，因此导数也叫微商。函数y=f(x)的微分又可记作dy=f′(x)dx。

梯度实际上是多元函数导数（或微分）的推广，用θ作为函数J(θ1​,θ2​,θ3​)的变量，其J(Θ)=0.55−(5θ1​+2θ2​−12θ3​)，则，其中△作为梯度的一种符号， ∂符号用于表示偏微分（部分的意思），即梯度就是分别对每个变量求偏微分，同时梯度用⟨⟩括起，表示梯度为一个向量。

梯度是微积分一个重要的概念，在单变量的函数中，梯度就是函数的微分（或对函数求导），代表函数在某个给定点切线的斜率；在多变量函数中，梯度是一个方向（向量的方向），梯度的方向指出了函数给定点上升最快的方向，而反方向是函数在给定点下降最快的方向。

导数是对含有一个自变量函数（一元）求导；偏导数是对含有多个自变量函数（多元）中的一个自变量求导。偏微分与偏导数类似，是对含有多个自变量函数（多元）中的一个自变量微分。

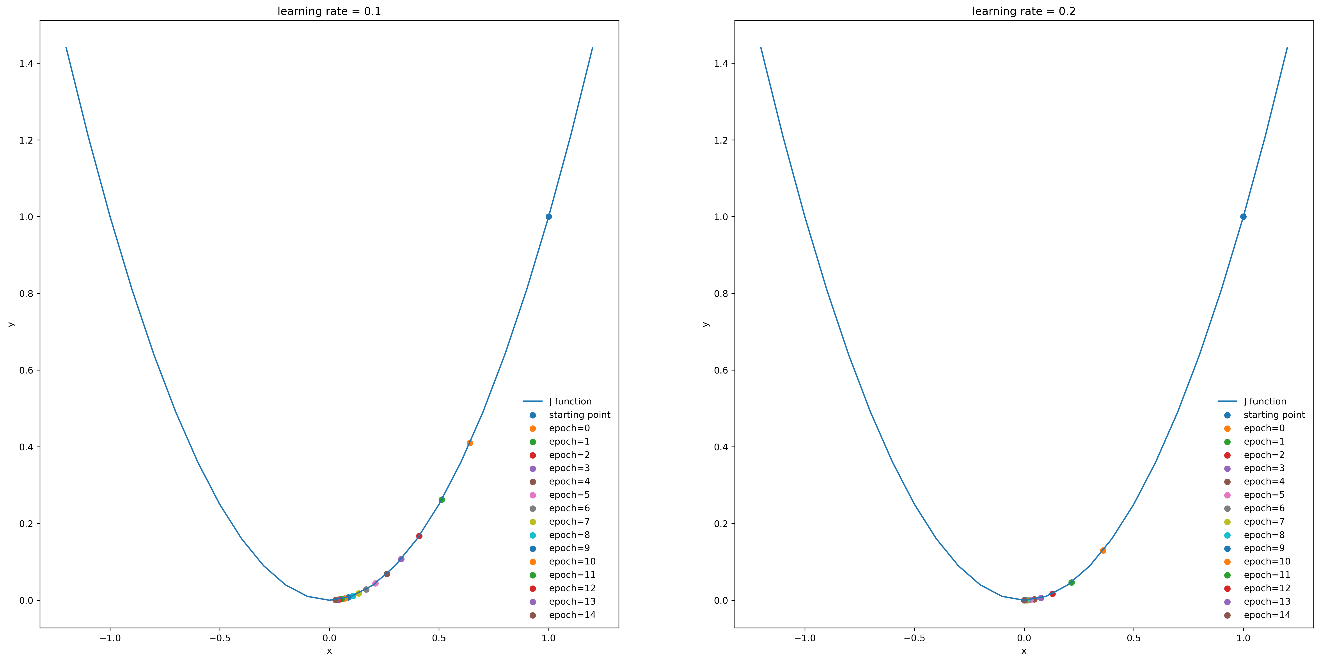
#### 2）梯度下降算法数学解释

公式为：，其中J是关于 Θ的函数，当前所处的位置为Θ0，要从这个点走到J(Θ)的最小值点Θ1，方向为梯度的反向，并用α（学习率或者步长，Learning rate or step size）超参数，控制每一步的距离，避免步子过大错过最低点，当愈趋近于最低点，学习率的控制越重要。梯度是上升最快的方向，如果往下走则需要在梯度前加−负号。

##### 单变量函数的梯度下降

定义函数为：，对该函数x微分，结果为：J′=2x，因此梯度下降公式为：，同时给定了迭代次数（iteration），通过打印图表可以方便查看每一迭代梯度下降的幅度。

import sympy  
import matplotlib.pyplot as plt  
import numpy as np  
  
# 定义单变量的函数，并绘制曲线  
x=sympy.symbols('x')  
J=1\*x\*\*2  
J\_=sympy.lambdify(x,J,"numpy")  
  
fig, axs=plt.subplots(1,2,figsize=(25,12),dpi=300)  
x\_val=np.arange(-1.2,1.3,0.1)  
axs[0].plot(x\_val,J\_(x\_val),label='J function')  
axs[1].plot(x\_val,J\_(x\_val),label='J function')  
  
# 函数微分  
dy=sympy.diff(J)  
dy\_=sympy.lambdify(x,dy,"math")  
  
# 初始化  
x\_0=1 # 初始化起点  
a=0.1 # 配置学习率  
iteration=15 # 初始化迭代次数  
  
axs[0].scatter(x\_0,J\_(x\_0),label='starting point')  
axs[1].scatter(x\_0,J\_(x\_0),label='starting point')  
  
# 根据梯度下降公式迭代计算  
for i in range(iteration):  
 if i==0:  
 x\_next=x\_0-a\*dy\_(x\_0)  
 x\_next=x\_next-a\*dy\_(x\_next)   
 axs[0].scatter(x\_next,J\_(x\_next),label='epoch=%d'%i)  
  
# 调整学习率，比较梯度下降速度  
a\_=0.2  
for i in range(iteration):  
 if i==0:  
 x\_next=x\_0-a\_\*dy\_(x\_0)  
 x\_next=x\_next-a\_\*dy\_(x\_next)   
 axs[1].scatter(x\_next,J\_(x\_next),label='epoch=%d'%i)  
   
axs[0].set(xlabel='x',ylabel='y')  
axs[1].set(xlabel='x',ylabel='y')  
axs[0].legend(loc='lower right', frameon=False)   
axs[1].legend(loc='lower right', frameon=False)   
  
axs[0].set\_title("learning rate = 0.1")  
axs[1].set\_title("learning rate = 0.2")  
plt.show()



上一部分代码是给定了回归系数的函数，其系数为1，通过梯度下降算法来查看梯度下降的趋势变化，需要注意的是上述定义的函数，是代表损失函数（可以写成，β是回归系数），不是模型（例如回归方程）。是由最初对损失函数求微分，并另结果为0，求回归系数，调整为在损失函数曲线上先求某点微分，找到该点的下降方向和大小（向量），并乘以学习率（调整下降速度），然后根据该向量移到下一个点，以此类推，直至找到下降趋势（梯度变化）趋近于0的位置，这个位置就是所要求的模型系数。

根据上述表述完成下述代码，即先定义模型，这个模型是回归模型，为了和上述损失函数的模型区别开来，定义其为：，并假设ω=3，来建立数据集，解释变量X，和响应变量y。定义模型的函数为model\_quadraticLinear(w,x)，使用SymPy库辅助表达和计算。有了模型之后，就可以根据模型和数据集计算损失函数，这里用MSE作为损失函数，计算结果为：。然后定义梯度下降函数，就是对MSE的ω求微分，加入学习率后计算结果为： G=0.0728520000000003w−0.218556000000001，准备好了这三个函数，就可以开始训练模型，顺序依次为：

1. 定义函数;
2. 定义损失函数;
3. 定义梯度下降;
4. 指定初始值，用损失函数计算残差平方和即MSE;
5. 比较MSE是否满足预设的精度’accuracy=1e-5’，如果不满足开始循环;
6. 由梯度下降公式计算下一个趋近于0的点，并再计算MSE，并比较MSE是否满足要求，周而复始;

计算结果为’w=3.008625’，约为3，正是最初用于生成数据集所假设的值。

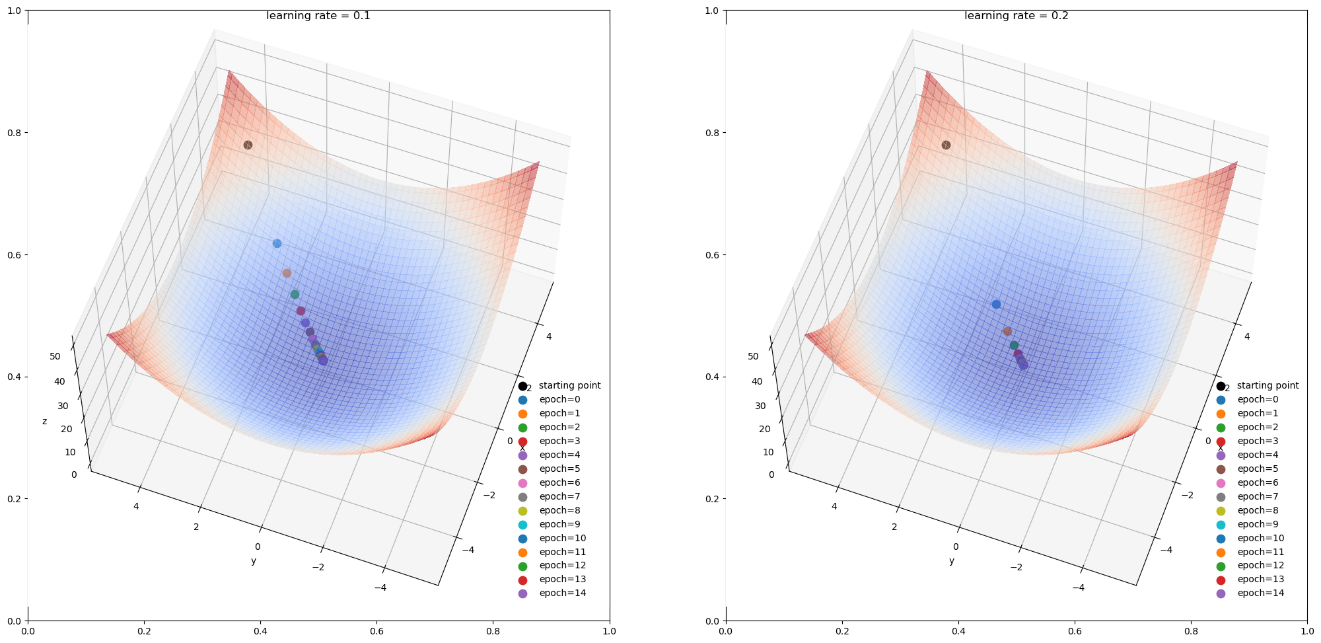
# 定义数据集，服从函数y= 3\*x\*\*2，方便比较计算结果  
import sympy  
from sympy import pprint  
import numpy as np  
  
x=sympy.symbols('x')  
y=3\*x\*\*2  
y\_=sympy.lambdify(x,y,"numpy")  
  
X=np.arange(-1.2,1.3,0.1)  
y=y\_(X)  
n\_size=X.shape[0]  
  
# 初始化  
a=0.15 # 配置学习率  
accuracy=1e-5 # 给出精度  
w\_=5 # 随机初始化系数  
  
# 定义模型  
def model\_quadraticLinear(w,x):  
 '''定义一元二次方程，不含截距b'''   
 return w\*x\*\*2  
  
# 定义损失函数  
def Loss\_MSE(model,X,y):  
 '''用均方误差（MSE）作为损失函数'''  
 model\_=sympy.lambdify(x,model,"numpy")  
 loss=(model\_(X)-y)\*\*2  
 return loss.sum()/n\_size/2  
   
# 定义梯度下降函数，是对损失函数求梯度  
def gradientDescent(loss,a,w):  
 '''定义梯度下降函数，即对模型变量微分'''  
 return a\*sympy.diff(loss,w)  
  
# 训练模型  
def train(X,y,a,w\_,accuracy):  
 '''根据精度值，训练模型'''  
 x,w=sympy.symbols(['x','w'])  
 model=model\_quadraticLinear(w,x)  
 print("定义函数：")  
 pprint(model)  
 loss=Loss\_MSE(model,X,y)  
 print("定义损失函数：")  
 pprint(loss)  
 grad=gradientDescent(loss,a,w)  
 print("定义梯度下降：")  
 pprint(grad)  
 print("\_"\*50)  
 grad\_=sympy.lambdify(w,grad,"math")  
 w\_next=w\_-grad\_(w\_)  
 loss\_=sympy.lambdify(w,loss,"math")  
 L=loss\_(w\_next)  
   
 i=0  
 print("迭代梯度下降，直至由损失函数计算的结果小于预设的值，w即为权重值（回归方程的系数）")  
 while not L<accuracy:  
 w\_next=w\_next-grad\_(w\_next)  
 L=loss\_(w\_next)  
 if i%10==0:   
 print("iteration:%d,Loss=%.6f,w=%.6f"%(i,L,w\_next))  
 i+=1  
 # if i%100:break  
 return w\_next  
  
w\_next=train(X,y,a,w\_,accuracy)

定义函数：  
 2  
w⋅x   
定义损失函数：  
 2   
1.75435200000001⋅(0.333333333333333⋅w - 1) + 0.431208⋅(0.333333333333333⋅w -   
  
 2  
1)   
定义梯度下降：  
0.0728520000000003⋅w - 0.218556000000001  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
迭代梯度下降，直至由损失函数计算的结果小于预设的值，w即为权重值（回归方程的系数）  
iteration:0,Loss=0.717755,w=4.719207  
iteration:10,Loss=0.158109,w=3.806898  
iteration:20,Loss=0.034829,w=3.378712  
iteration:30,Loss=0.007672,w=3.177746  
iteration:40,Loss=0.001690,w=3.083424  
iteration:50,Loss=0.000372,w=3.039154  
iteration:60,Loss=0.000082,w=3.018377  
iteration:70,Loss=0.000018,w=3.008625

##### 2. 多变量函数的梯度下降

多变量函数的梯度下降与单变量函数的梯度下降类似，只是在求梯度时是分别对各个变量求梯度，两者之间不互相干扰。计算结果如下。

import sympy  
import matplotlib.pyplot as plt  
from matplotlib import cm  
import numpy as np  
  
# 定义单变量的函数，并绘制曲线  
x1,x2=sympy.symbols(['x1','x2'])  
J=x1\*\*2+x2\*\*2  
J\_=sympy.lambdify([x1,x2],J,"numpy")  
  
x1\_val=np.arange(-5,5,0.1)  
x2\_val=np.arange(-5,5,0.1)  
x1\_mesh,x2\_mesh=np.meshgrid(x1\_val,x2\_val)  
y\_mesh=J\_(x1\_mesh,x2\_mesh)  
  
fig, axs=plt.subplots(1,2,figsize=(25,12))  
axs[0]=fig.add\_subplot(1,2,1, projection='3d')  
surf=axs[0].plot\_surface(x1\_mesh,x2\_mesh ,y\_mesh, cmap=cm.coolwarm,linewidth=0, antialiased=False,alpha=0.5,)  
axs[1]=fig.add\_subplot(1,2,2, projection='3d')  
surf=axs[1].plot\_surface(x1\_mesh,x2\_mesh ,y\_mesh, cmap=cm.coolwarm,linewidth=0, antialiased=False,alpha=0.5,)  
# fig.colorbar(surf, shrink=0.5, aspect=5)  
  
# 函数x1,x2微分  
dx1=sympy.diff(J,x1)  
dx2=sympy.diff(J,x2)  
dx1\_=sympy.lambdify(x1,dx1,"math")  
dx2\_=sympy.lambdify(x2,dx2,"math")  
  
# 初始化  
x1\_0=4 # 初始化x1起点  
x2\_0=4 # 初始化x2起点  
iteration=15 # 初始化迭代次数  
a=0.1 # 配置学习率  
  
axs[0].scatter(x1\_0,x2\_0,J\_(x1\_0,x2\_0),label='starting point',c='black',s=80)  
axs[1].scatter(x1\_0,x2\_0,J\_(x1\_0,x2\_0),label='starting point',c='black',s=80)  
  
# 根据梯度下降公式迭代计算  
for i in range(iteration):  
 if i==0:  
 x1\_next=x1\_0-a\*dx1\_(x1\_0)  
 x2\_next=x2\_0-a\*dx2\_(x2\_0)  
   
 x1\_next=x1\_next-a\*dx1\_(x1\_next)   
 x2\_next=x2\_next-a\*dx2\_(x2\_next)   
 axs[0].scatter(x1\_next,x2\_next,J\_(x1\_next,x2\_next),label='epoch=%d'%i,s=80)  
   
# 调整学习率，比较梯度下降速度  
a\_=0.2  
for i in range(iteration):  
 if i==0:  
 x1\_next=x1\_0-a\_\*dx1\_(x1\_0)  
 x2\_next=x2\_0-a\_\*dx2\_(x2\_0)  
   
 x1\_next=x1\_next-a\_\*dx1\_(x1\_next)   
 x2\_next=x2\_next-a\_\*dx2\_(x2\_next)   
 axs[1].scatter(x1\_next,x2\_next,J\_(x1\_next,x2\_next),label='epoch=%d'%i,s=80)  
  
   
axs[0].set(xlabel='x',ylabel='y',zlabel='z')  
axs[1].set(xlabel='x',ylabel='y')  
axs[0].legend(loc='lower right', frameon=False)   
axs[1].legend(loc='lower right', frameon=False)   
  
axs[0].view\_init(60,200) # 可以旋转图形的角度，方便观察  
axs[1].view\_init(60,200)  
axs[0].set\_title("learning rate = 0.1")  
axs[1].set\_title("learning rate = 0.2")  
plt.show()

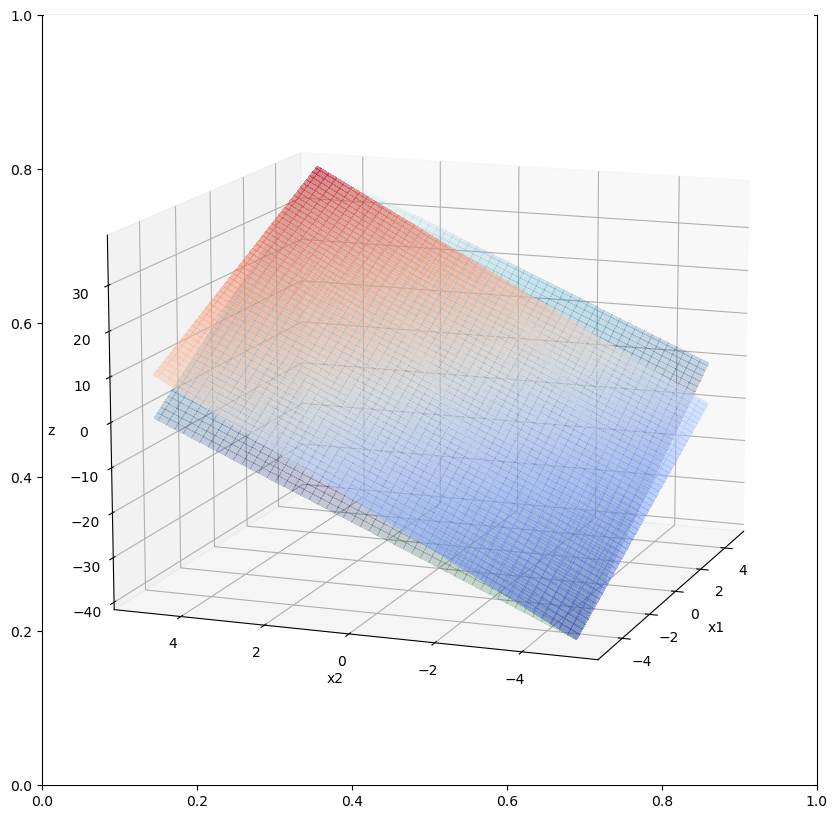


用梯度下降方法求解二元函数模型，其过程基本同上述求解一元二次函数模型，注意在下述求解过程中，学习率的配置对计算结果有较大影响，可以尝试不同的学习率观察所求回归系数的变化。计算结果在α=0.01的条件下， w=3.96， v=4.04，与假设的值3,5还是有段距离，也可以打印图形，观察真实平面与训练所得模型的平面之间的差距。可以正则化，即增加惩罚项L2​或L1​尝试改进。不过这已经能够对梯度下降算法有个比较清晰的理解，这是后续应用sklearn机器学习库和[PyTorch](https://pytorch.org/)③深度学习库非常重要的基础。

同时，也应用了sklearn库提供的SGDRegressor（ Stochastic Gradient Descent）随机梯度下降方法训练该数据，其w=2.9999586， v=4.99993523，即约为3和5,与原假设的系数值一样。

# 定义数据集，服从函数y= 3\*x\*\*2，方便比较计算结果  
import sympy  
from sympy import pprint  
import numpy as np  
x1,x2=sympy.symbols(['x1','x2'])  
y=3\*x1+5\*x2  
y\_=sympy.lambdify([x1,x2],y,"numpy")  
  
X1\_val=np.arange(-5,5,0.1)  
X2\_val=np.arange(-5,5,0.1)  
y\_val=y\_(X1\_val,X2\_val)  
n\_size=y\_val.shape[0]  
  
# 初始化  
a=0.01 # 配置学习率  
accuracy=1e-10 # 给出精度  
w\_=5 # 随机初始化系数，对应x1系数  
v\_=5 # 随机初始化系数，对应x2系数  
  
# 定义模型  
def model\_quadraticLinear(w,v,x1,x2):  
 '''定义二元一次方程，不含截距b'''   
 return w\*x1+v\*x2  
  
# 定义损失函数  
def Loss\_MSE(model,X1,X2,y):  
 '''用均方误差（MSE）作为损失函数'''  
 model\_=sympy.lambdify([x1,x2],model,"numpy")  
 loss=(model\_(x1=X1,x2=X2)-y)\*\*2  
 return loss.sum()/n\_size/2  
  
# 定义梯度下降函数，是对损失函数求梯度  
def gradientDescent(loss,a,w,v):  
 '''定义梯度下降函数，即对模型变量微分'''  
 return a\*sympy.diff(loss,w),a\*sympy.diff(loss,v)  
  
# 训练模型  
def train(X1\_val,X2\_val,y\_val,a,w\_,v\_,accuracy):  
 '''根据精度值，训练模型'''  
 w,v=sympy.symbols(['w','v'])  
 model=model\_quadraticLinear(w,v,x1,x2)  
 print("定义函数：")  
 pprint(model)  
 loss=Loss\_MSE(model,X1\_val,X2\_val,y\_val)  
 print("定义损失函数：")  
 pprint(loss)   
 grad\_w,grad\_v=gradientDescent(loss,a,w,v)  
 print("定义梯度下降：")  
 pprint(grad\_w)  
 pprint(grad\_v)  
 print("\_"\*50)   
 gradw\_=sympy.lambdify([v,w],grad\_w,"math")  
 gradv\_=sympy.lambdify([v,w],grad\_v,"math")  
   
 w\_next=w\_-gradw\_(v\_,w\_)   
 v\_next=v\_-gradv\_(v\_,w\_)  
 loss\_=sympy.lambdify([v,w],loss,"math")  
 L=loss\_(w=w\_next,v=v\_next)   
   
 i=0  
 print("迭代梯度下降，直至由损失函数计算的结果小于预设的值，w,v即为权重值（回归方程的系数）")  
 while not L<accuracy:   
 w\_next=w\_next-gradw\_(w=w\_next,v=v\_next)  
 v\_next=v\_next-gradv\_(v=v\_next,w=w\_next)  
 L=loss\_(w=w\_next,v=v\_next)   
 if i%10==0:   
 print("iteration:%d,Loss=%.6f,w=%.6f,v=%.6f"%(i,L,w\_next,v\_next))  
 i+=1  
 return w\_next,v\_next  
  
w\_next,v\_next=train(X1\_val,X2\_val,y\_val,a,w\_,v\_,accuracy)  
  
  
fx=w\_next\*x1+v\_next\*x2  
fx\_=sympy.lambdify([x1,x2],fx,"numpy")  
fig, ax=plt.subplots(figsize=(10,10))  
ax=fig.add\_subplot( projection='3d')  
  
x1\_mesh,x2\_mesh=np.meshgrid(X1\_val,X2\_val)  
y\_mesh=y\_(x1\_mesh,x2\_mesh)  
y\_pre\_mesh=fx\_(x1\_mesh,x2\_mesh)  
  
surf=ax.plot\_surface(x1\_mesh,x2\_mesh ,y\_mesh, cmap=cm.coolwarm,linewidth=0, antialiased=False,alpha=0.5,)  
surf=ax.plot\_surface(x1\_mesh,x2\_mesh ,y\_pre\_mesh, cmap=cm.ocean,linewidth=0, antialiased=False,alpha=0.2,)  
# fig.colorbar(surf, shrink=0.5, aspect=5)  
  
# 用sklearn库提供的SGDRegressor（ Stochastic Gradient Descent）随机梯度下降方法训练  
from sklearn.linear\_model import SGDRegressor  
from sklearn.model\_selection import train\_test\_split  
  
X\_=np.stack((x1\_mesh.flatten(),x2\_mesh.flatten())).T  
y\_=y\_mesh.flatten()  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_, y\_, test\_size=0.33, random\_state=42)  
SDGreg=SGDRegressor(loss='squared\_loss') # 配置损失函数，其正则化，即惩罚项默认为L2  
SDGreg.fit(X\_train,y\_train)  
print("\_"\*50)  
print("Sklearn SGDRegressor test set r-squared score%s"%SDGreg.score(X\_test,y\_test))  
print("Sklearn SGDRegressor coef\_:",SDGreg.coef\_)  
  
ax.set(xlabel='x1',ylabel='x2',zlabel='z')  
ax.view\_init(13,200) # 可以旋转图形的角度，方便观察  
plt.show()

定义函数：  
v⋅x₂ + w⋅x₁  
定义损失函数：  
 2   
137.360000000001⋅(-0.125⋅v - 0.125⋅w + 1) + 129.359999999998⋅(0.125⋅v + 0.125  
  
 2  
⋅w - 1)   
定义梯度下降：  
0.0833499999999994⋅v + 0.0833499999999994⋅w - 0.666799999999996  
0.0833499999999994⋅v + 0.0833499999999994⋅w - 0.666799999999996  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
迭代梯度下降，直至由损失函数计算的结果小于预设的值，w,v即为权重值（回归方程的系数）  
iteration:0,Loss=8.172455,w=4.694389,v=4.705967  
iteration:10,Loss=0.251475,w=4.091926,v=4.153720  
iteration:20,Loss=0.007738,w=3.986244,v=4.056846  
iteration:30,Loss=0.000238,w=3.967706,v=4.039853  
iteration:40,Loss=0.000007,w=3.964454,v=4.036872  
iteration:50,Loss=0.000000,w=3.963883,v=4.036349  
iteration:60,Loss=0.000000,w=3.963783,v=4.036258  
iteration:70,Loss=0.000000,w=3.963766,v=4.036241  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Sklearn SGDRegressor test set r-squared score0.9999999998894413  
Sklearn SGDRegressor coef\_: [2.99996858 4.99994742]



#### 3）用Sklearn库提供的SGDRegressor（ Stochastic Gradient Descent）随机梯度下降方法训练公共健康数据

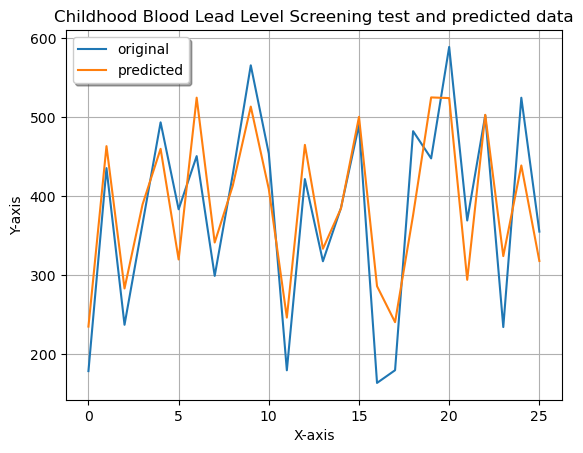
使用SGDRegressor随机梯度下降训练多元回归模型，参数设置中损失函数配置为’squared\_loss’，惩罚项(penalty)默认为L2（具体信息可以从Sklearn官网获取）。计算结果中决定系数为0.76，较多项式回归偏小。计算获取6个系数，对应6个特征。

from sklearn.linear\_model import SGDRegressor  
from sklearn.model\_selection import train\_test\_split  
from sklearn.pipeline import make\_pipeline  
  
X\_=data\_Income[['Per Capita Income','Below Poverty Level','Crowded Housing','Dependency','No High School Diploma','Unemployment',]].to\_numpy() # 将特征值数据格转换为numpy的特征向量矩阵  
y\_=data\_Income['Childhood Blood Lead Level Screening'].to\_numpy() # 将目标值数据格式转换为numpy格式的向量  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_, y\_, test\_size=0.33, random\_state=42)  
SDGreg=make\_pipeline(StandardScaler(),  
 SGDRegressor(loss='squared\_loss',  
 max\_iter=1000,   
 tol=1e-3))  
SDGreg.fit(X\_train,y\_train)  
print("\_"\*50)  
print("Sklearn SGDRegressor test set r-squared score%s"%SDGreg.score(X\_test,y\_test))  
print("Sklearn SGDRegressor coef\_:",SDGreg[1].coef\_)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Sklearn SGDRegressor test set r-squared score0.7622218232194027  
Sklearn SGDRegressor coef\_: [ 3.09516933 36.26810545 27.85043992 5.83100125 42.48193716 15.37895748]

图表打印查看测试数据集观测值和预测值的分布情况。从结果来看，预测值基本趋向于观测值，结合决定系数为0.76，可以判断训练的模型具有较好的预测表现能力。

y\_pred=SDGreg.predict(X\_test)  
  
x\_ax=range(len(y\_test))  
plt.plot(x\_ax, y\_test, label="original")  
plt.plot(x\_ax, y\_pred, label="predicted")  
plt.title("Childhood Blood Lead Level Screening test and predicted data")  
plt.xlabel('X-axis')  
plt.ylabel('Y-axis')  
plt.legend(loc='best',fancybox=True, shadow=True)  
plt.grid(True)  
plt.show()



关于损失函数，代价函数/成本函数

损失函数（Loss Function），针对单个样本，衡量单个样本的预测值和观测值之间的差距；

代价函数/成本函数（Cost Function），针对多个样本，衡量多个样本的预测值和观测值之间的差距；

实际上，对于这三者的划分在实际相关文献使用上并没有体现出来，往往混淆，因此也不做特殊界定。

注释（Notes）：

① scikit-learn，也称为sklearn，针对Python编程语言的免费软件机器学习库（<https://scikit-learn.org/stable/index.html>）。

② PySAL，支持空间数据科学计算的开源项目（<https://pysal.org/>）。

③ PyTorch，一个基于Torch库的开源机器学习框架，用于计算机视觉和自然语言处理等应用（<https://pytorch.org/>）。

参考文献（References）:

[1] Cavin Hackeling.张浩然译.scikit-learning 机器学习[M].人民邮电出版社.2019.2.（ Cavin Hackeling. Mastering Machine Learning with scikit-learn[M].Packt Publishing Ltd.July 2017.Second published.）

[2] Gradient Descent - Problem of Hiking Down a Mountain. Udacity (<https://jums.club/pdf/Gradient_Descent.pdf>).