**几种重要的非线性回归模型**

## 【方法介绍】

考虑这样的一个问题: 给定一系列样本: , 其中, 是一未知函数, 参数可能具有无限维. 求的最佳估计.

这是非参数回归问题的一般形式. 本项目将介绍两种非参数回归的常用方法: 局部多项式回归和样条回归, 并在最后分析它们与参数回归的基本方法最小二乘法之间的差异.

**1.局部多项式回归介绍**

顾名思义, 局部多项式回归的主要思想是利用与当前点邻近的点所对应的, 对进行估计. 具体做法如下:

写出在点处的泰勒级数: . 取级数的前项, . 一般取0,1,2,3.

现在要做的就是写出损失函数. 局部多项式回归在这一步上引入了权重的概念: 对周围每个点的损失指派一个权重, 权重分布满足, 并且距离越近的点, 权重越大; 反之权重越小. 基于此, 局部多项式回归的损失函数可以写为: .

式中, ,, .

对求偏导来最小化损失函数, 可得. 于是对每个样本点的估计值.

由上述局部多项式回归的过程, 可以发现其有两个超参数: 拟合多项式的最高次数和权重. 下面分别对这两个参数的选取进行讨论:

**i)权重w**

权函数显然是一个关于距离的函数, 理论上任何满足之前所述性质的函数都能作为权函数. 在局部线性回归中, 权重通常采用核函数来度量. 这里的权函数概念不同于支持向量机中的核函数, 它可以理解为是一种概率密度函数. 一般将核函数记为. 常见的核函数有: Gaussian核, Epanechnikov核, tricube核等.

将核函数用于局部线性回归, 则权重. 式中称为窗宽, 也是一个超参数.

窗宽度量了局部多项式回归模型的复杂度. 越大的窗宽, 会有越多的样本点用于估计, 指派到每个样本点的权重也会较小, 导致近样本点与远样本点的权重差距不那么大, 因此这样得到的拟合曲线也会越光滑, 但同样如前所述, 估计的偏差会相对变大, 因为利用了大量无关局域性质的样本点的信息; 而较小的窗宽与之相反, 拟合的方差会因偏差的减小而增大, 模型泛化误差较大.

因此一个好的窗宽必须达到经验误差与泛化误差之间的平衡. 在优化调参的时候, 可以采用交叉验证或施加一定的惩罚来提高模型的泛化能力. 比如可以证明, 在使用高斯核函数的情况下最好的窗宽为, 其中是样本标准差. 另外, 如果在不同区域拟合的时候, 采用不同的窗宽, 也不失为一种选择, 特别是当样本点的密集程度在不同区间有明显差异时, 这种可变窗宽的优势会更明显. 下图是大窗宽与小窗宽的拟合效果对比:

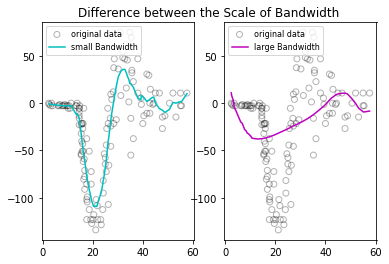


图1 窗宽大小对回归的影响

核函数决定了权重指派的具体方式. 不同的核函数会改变较远样本点与较近样本点权重指派的相对差异, 对回归结果的影响并不明显. 经过几次尝试, 发现较为复杂的核函数可能会使回归曲线变得更为粗糙, 而简单的核函数比如, 回归曲线则相对平滑. 以下是采用高斯核函数和采用简单核函数的拟合效果对比:

图表, 散点图

描述已自动生成

图2 核函数对回归效果的影响

可以看到, 两者虽有区别, 但是显然没有窗宽对模型的影响大.

**ii)最高次数d**

当时, 设计矩阵退化为维全1向量, 因此. 可见在这种情形下, 只是用附近的点的值来估计. 这也是此时多项式回归被成为核平滑的原因.

当时, , 此时的局部多项式回归被称为局部线性回归. 另外, 当窗宽时, 会用所有的样本点来估计, 因此也可以把局部线性回归看成是线性回归的合理外推. 在某些具有明显线性关系的区域内, 局部线性回归具有极好的性能, 并且相比于的核平滑方式, 在样本点的边界区域, 局部线性回归的偏差较小, 因为它不仅仅利用了响应变量的信息, 还利用了估计点与邻近样本点之间的线性关系, 一定程度上减小了偏差. 下图展示了这两种方式在边界区域的性能（样本由加上高斯噪声产生）, 显然核平滑在边界处产生了明显的偏差（在左边界受到右侧数据影响上偏, 右边界受到左侧数据影响下偏）, 而局部线性回归则几乎没有受到影响.

图表, 散点图

描述已自动生成

图3 两者在边界处的性能对比

局部线性回归的偏差和方差分别为: . 式中是样本容量, 是窗宽. 由此同样可以说明, 当窗宽减小时, 估计值的方差会迅速增加; 而较大的窗宽又会引起较大的经验误差.

当时, 估计方程变为抛物线型, 它在一些具有“峰”或“谷”的位置呈现较好的拟合效果, 而核平滑与局部线性回归在这些位置的估计值均会产生较大偏差.

最高次数与窗宽一样, 也是用来度量模型复杂度的参数. 一般在局部多项式回归中, 最高不会超过3. 即使样本十分复杂, 也可以通过设置节点分割区间, 在每一个小区间内再使用局部多项式回归的方法, 减小模型复杂度.

综上所述, 局部多项式回归是非参数回归中常用的方法, 它综合了多项式回归和线性加权回归的优点, 通过局部加权的方式, 用邻近样本点的信息来对当前数据点进行估计, 对非线性模型具有较好的回归性能.

**2.样条回归介绍**

样条回归与局部多项式回归一样, 都是非参数回归的重要方法. 现在假设样本点有很明显的分段特征, 那么自然想到可以利用分段函数来估计. 样条回归便是采取了分段函数这一策略, 并且加上了一个限制条件: 这族分段函数必须是连续函数.

首先给出最简单的样条回归模型线性样条回归: .

式中, 有序变量被称为控制节点, 为模型待估计的参数, 记号.

可以清楚的看到, 每经过一个控制节点, 估计量中就会加入一个线性项, 从而使模型能拟合控制节点之后的区间. 这就是线性样条的原理. 至于参数的估计, 同样采用最小二乘法最小化损失函数来求得: .

这时的设计矩阵为: .

将它合理外推可以得到阶样条: . 类似于线性样条要求函数连续, 对阶样条有要求阶导数连续. 因此, 高阶样条会更光滑, 具有更好的预测能力. 在这些高阶样条中, 最为常用的是三阶样条模型.

样条回归的一个重要概念是自由度, 可以理解为模型参数的维度, 越高的自由度会带来越高的灵活性. 对于一个具有*K*个控制节点的三阶样条, 其自由度为*K+*4, 也即参数***w***的维度为*K+*4. 通过增加控制节点的个数, 可以很好的降低模型的经验误差, 但模型的解释能力会变差.

为了提高模型对控制节点以外区域的预测能力, 一种可行的办法是使用自然样条. 自然样条是在原有样条的基础上, 限制模型在控制节点以外的高阶（二阶及以上）导数值为0; 具体而言, 自然样条规定模型在控制节点以外区域使用线性模型进行预测, 使得这些区域的预测值符合实际预期. 下面是自然样条的数学推导过程（以自然三阶样条为例）:

三阶样条的模型为. 现在令. 经过计算可以得到, 的系数需要满足. 这就是自然三阶样条的数学模型.

另外一种方法类似于线性回归方法中常用的岭回归和Lasso. 越多的控制节点会带来越低的经验误差, 因此可以首先设定大量的控制节点, 再设置一个罚函数用于平衡节点数量, 从而达到经验误差与泛化误差的平衡. 这种带惩罚项的样条模型被称为平滑样条. 由以上讨论可知, 平滑样条是一个最小化问题:



式中是大于0的常数, 决定了对复杂模型的惩罚力度; 在实际应用时, 可以采用留一交叉验证或者广义交叉验证来选择出最佳惩罚系数.

但是平滑样条往往趋于选择较少的控制节点. 若将上式做一些简化, 改变一下罚函数的形式:

, 其中

上式的模型就是带惩罚的样条回归, 相比平滑样条, 它的罚函数变为模型参数***w***的2范数的平方, 于是可以像岭回归那样直接求出***w***的最小二乘估计. 它结合了普通样条和平滑样条的优点, 既有较高的灵活性, 又有较强的泛化能力.

**3.样条与最小二乘、局部线性回归之间差异的分析**

就这三者的应用领域而言, 样条和局部线性回归都属于非参数回归, 而最小二乘法则是参数回归中估计待定参数的重要方法.

非参数回归在最开始已经做过介绍, 它最大特点是解释变量与响应变量之间的真实模型未知, 而不同于普通最小二乘回归等参数回归只需要对真实模型的参数做出估计. 先简要介绍一下最小二乘回归:

最小二乘回归的方法为: 给出一系列基函数, 首先假定真实模型为这些基函数的线性组合: . 之后采用最小二乘法最小化残差平方和:

.

式中. 令

, 得待求参数估计.

很多情况下, 基函数通常取为一系列幂函数, 即, 这种回归也被称为多项式回归. 但是当样本点之间的关系十分复杂, 离散程度很不均匀时, 多项式回归的效果并不好. 具体分析如下:

根据预测误差、噪音、偏差、方差之间的关系（预测误差=噪音+偏差+方差）:

,

若采取增加基函数的策略（例如使用六次、七次甚至更高次方的多项式来拟合）来提高拟合水平, 虽然MSE会显著降低, 但是拟合曲线的方差会非常大, 每增加一个样本都会引起曲线的剧烈波动, 也即发生过拟合, 模型的泛化能力非常弱; 而若仍然采用较低次数的多项式来拟合, 则会出现另一种极端——欠拟合, 即估计值虽然有较小的方差, 曲线较为平滑, 但是距离真实响应变量比较远, 这时偏差成为了曲线拟合的主要误差来源. 因此拟合多项式最高次数的选择较为困难.

这便是最小二乘法的一个重要缺陷所在, 即模型泛化能力弱, 如果向样本中添加新样本或是从样本中取出一些样本, 都会导致回归曲线的剧烈波动, 有时甚至需要直接更换模型假设. 这也引出了最小二乘法的另一个不足: 回归函数的模型需要预先假定; 但是单凭经验假定的模型往往缺乏稳健性.

而回归样条、局部线性回归等非参数回归策略, 不仅对样本的分布没有过多的假设, 而且由于并不给出具体回归函数的形状, 具有较强的适应性, 模型完全由样本数据来确定, 因而也具有较高的灵活度. 在非线性、非齐次模型的回归上, 它们相比最小二乘会有明显的优势.

但是, 也正是因为非参数回归的参数灵活多变, 它也容易陷入“维数灾难”, 比如平滑样条中惩罚系数的选取, 直接影响到回归曲线的光滑程度、对样本的拟合程度, 一旦失准就会造成过拟合或欠拟合. 除此之外, 由于样条、局部线性回归等没有给出具体的真实模型, 它们无法像最小二乘那样进行外推运算; 且估计精度很依赖样本的大小, 只有样本容量足够大时才会有较高的精度.

至于样条和局部线性回归之间的差异, 主要是源于两者采用的数学近似不同. 样条使用分段函数进行估计: 在两个控制节点之间使用同一函数得到估计值, 在两个不同的控制区域则使用不同的函数来进行估计. 于是对样条回归效果影响最大的是控制节点数量和位置的选取. 而局部线性回归利用的是真实模型的泰勒级数, 即用预测点邻近的各点来估计真实值. 具体距离多近才算是“邻近”, 则采用窗宽来度量, 它便成为了局部线性回归中影响力最大的超参数.

这些底层逻辑上的差异, 使得样条曲线比局部线性回归更为平滑, 自由度更高; 但是它也是更不稳定的回归方法, 在控制节点以外的区域有时甚至会比最小二乘回归产生更大的泛化误差. 在处理不同问题时, 两种方法应斟酌选用.

## 【案例实战】

## 案例一:

图表, 散点图

描述已自动生成

图4 mcycle dataset

观察数据集, 可以发现自变量time和因变量head acceleration之间有明显的非线性关系; 因此需要采用非线性回归来解释. 对于本案例, 将采用四种非线性回归的方法: 线性样条、三阶样条、局部线性回归、局部二次多项式回归. 之后从经验误差和泛化误差两个角度综合考量, 选择出四个模型中最优的模型, 并对其进行调参, 达到最佳的模型性能.

以下是本案例的实践步骤:

## 步骤一: 读入数据观察其特征

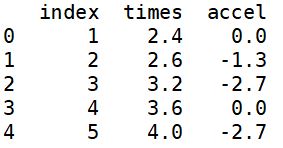
data = pd.read\_csv("C:\\Users\\16548\\Desktop\\导论作业\\mcycle.csv")

x = data["times"]

y = data["accel"]

# 导入数据, times是解释变量, accel是响应变量

数据导入后, 可以发现数据集中有133个样本, 两个特征. 使用命令data.head(5)观察data前五个样本:



样本只有1个解释变量times和1个响应变量accel, 并且通过data.isnull()命令发现数据集中没有缺失数据. 考虑作出散点图如下:

图表, 散点图

描述已自动生成

图5 数据散点图

显然times和accel之间是非线性关系, 需要采用一些非线性回归的技术对真实模型进行估计.

## 步骤二: 构建四个非线性回归模型

常用的非线性回归模型有: 核平滑、样条回归、局部多项式回归等. 这里选用其中四种模型, 分别是线性样条、三阶样条、局部线性回归、局部多项式回归. 它们的代码如下:

**线性样条:**

def LinearSpineRegression(x, y):  # 简单线性样条回归

    n = len(x)

    x = np.matrix(x).T  # x是解释变量

    y = np.matrix(y).T  # y是因变量

    G = np.matrix(np.ones(n)).T  # G是设计矩阵

    G = np.concatenate((G, x), axis=1)

    knots = np.array([12, 23, 32, 38])  # knots是样条的节点

    for i in knots:

        tmp = np.matrix(x)

        for j in range(n):

            if tmp[j] < i:

                tmp[j] = 0

            else:

                tmp[j] = tmp[j] - i

        G = np.concatenate((G, tmp), axis=1)

# 构造设计矩阵G

    beta = (G.T \* G).I \* G.T \* y  # 采用最小二乘计算参数beta

    yr = []

    for i in range(n):

        s = beta[0] + beta[1] \* x[i]

        for j in range(4):

            tmp = 0

            if x[i] > knots[j]:

                tmp = x[i] - knots[j]

            s = s + beta[j + 2] \* tmp

        s = float(s)

        yr.append(s)

return yr  # 返回回归值yr

**三阶样条:**

def CubicSplineRegression(

    train\_x, train\_y, valid\_x, y\_hat, knots

):  # 三阶样条回归, train\_x, train\_y是训练集; valid\_x是验证集, y\_hat记录测试集回归值, knots是节点

    n = len(train\_x)

    train\_x = np.matrix(train\_x).T

    train\_y = np.matrix(train\_y).T

    G = np.matrix(np.ones(n)).T  # G是设计矩阵

    for i in range(3):

        G = np.concatenate((G, np.power(train\_x, i + 1)), axis=1)

    for i in knots:

        tmp = np.matrix(train\_x)

        for j in range(n):

            if tmp[j] < i:

                tmp[j] = 0

            else:

                tmp[j] = tmp[j] - i

        G = np.concatenate((G, np.power(tmp, 3)), axis=1)

# 构造设计矩阵G

    beta = (G.T \* G).I \* G.T \* train\_y  # 采用最小二乘计算参数beta

    # 估计参数beta

    n = len(valid\_x)

    valid\_x = np.matrix(valid\_x).T

    for i in range(n):

        s = (

            beta[0]

            + beta[1] \* valid\_x[i]

            + beta[2] \* valid\_x[i] \*\* 2

            + beta[3] \* valid\_x[i] \*\* 3

        )

        for j in range(4):

            tmp = 0

            if valid\_x[i] > knots[j]:

                tmp = valid\_x[i] - knots[j]

            s = s + beta[j + 4] \* tmp\*\*3

        s = float(s)

        y\_hat.append(s)

    # 计算验证集的估计值y\_hat

return y\_hat  # 返回回归值y\_hat

**局部多项式回归（局部线性回归是局部多项式回归的一个情形, 代码可以重用）:**

def LocalPolynomialRegression(x, y, h, d):  # 局部多项式回归, x是解释变量, y是响应变量, h是窗宽, d是多项式的次数（为1时表示局部线性回归）

    n = len(x)

    X = np.matrix(np.ones(n)).T

    Dmatrix = np.matrix(np.ones(n)).T

    y = np.array(y).reshape(len(y), 1)

    yr = []

    for i in range(n):

        for j in range(1, d + 1):

            xpow = np.matrix(np.power((x - x[i]), j)).T

            Dmatrix = np.concatenate((X, xpow), axis=1)  # 构造设计矩阵Dmatrix

        local\_index = []  # local\_index用来记录与点x[i]邻近的点的索引

        for j in range(n):

            if np.abs(x[j] - x[i]) <= h:

                local\_index.append(j)

        Dmatrix\_ = Dmatrix[local\_index]

        y\_ = y[local\_index]

        Weight = np.exp(-((x - x[i]) \*\* 2) / (2 \* h\*\*2)) / (

            np.sqrt(2 \* np.pi) \* h

        )  # 这里权函数采用高斯核函数

        Weight = Weight[local\_index]

        W = np.diag(Weight)

        beta = ((Dmatrix\_.T \* W \* Dmatrix\_).I) \* (

            Dmatrix\_.T \* W \* y\_

        )  # 同样可以采用最小二乘得到参数beta

        yr.append(float(beta[0]))  # 回归值用yr记录

return yr

## 步骤三: 利用所建模型进行回归并选择最佳模型

分别调用步骤二中已经定义的函数, 可以得到四个回归结果yr1、yr2、yr3和yr4:

yr1 = LinearSpineRegression(x, y)

# 简单线性样条回归的结果yr1

yr2 = CubicSplineRegression(x, y, x, [], [15, 20, 32, 40])

# 三阶样条回归的结果yr2

yr3 = LocalPolynomialRegression(x, y, 5, 1)

# 局部线性回归的结果yr3

yr4 = LocalPolynomialRegression(x, y, 10, 2)

# 局部多项式回归（次数为2）的结果yr4

分别画出四条回归曲线:

图表, 折线图

描述已自动生成

图6 线性样条和三阶样条的回归结果

图表, 折线图

描述已自动生成

图7 局部线性回归和局部二次多项式回归结果

从这四者的回归曲线可以看出, 线性样条的回归效果最差, 直观上看像是五段直线方程在控制节点处“拼接”而成, 几乎不能解释模型.

两种局部回归模型的曲线显得相对平滑, 也基本能体现样本点的特征. 具体来分析, 在所有数据的左侧, 局部线性回归的拟合效果要优于局部二次多项式回归, 这可能是因为该区域数据呈明显的线性关系, 且几乎是一个常数值, 而局部二次多项式强行给予该处样本点平方关系, 产生了较大的偏差; 此外, 在times约为21的区域, 二者估计值因该处数据极为剧烈的下降和上升都产生了偏差, 但二次多项式由于在“峰”和“谷”区域较好的估计效果, 它的偏差会略小于局部线性回归; 而在其他区域, 它们的估计效果大都可以接受.

三次样条在这四种模型中的表现最为优秀. 整条回归曲线十分平滑, 对其他三个模型估计较差的区域也有很好的拟合效果.

再来观察回归的RMSE情况:

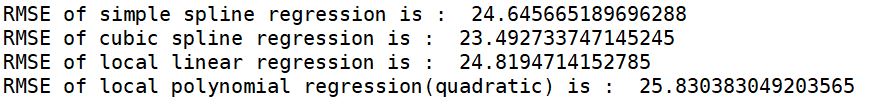


图8 四种回归模型的RMSE

四种回归方法的残差都很接近, 相比之下也是三次样条的RMSE最小. 因此接下来将对三次样条进行调参, 获得最佳拟合模型.

## 步骤四: 对三阶样条进行调参优化

三阶样条的超参数是控制节点的数量及其位置. 从数据集的角度观察, 将控制节点的数量设置为4应该较为准确, 这四个位置分别是散点图中有明显分段的四个点. 而且可以知道, 第一个节点位置的大致范围为: , 第二个大致为: , 第三个大致为: , 第四个大致为: . 这已经大大简化了原本的调参问题. 接下来只需要在这四个范围内进行枚举搜索最佳参数即可.

在搜索过程中, 本案例采用了类似坐标下降的方法, 而不是使用四重循环一个个进行枚举,提高了一定的效率. 即首先固定其他三个节点的位置, 对第一个节点进行下降, 将得到的值作为第一个节点的最优值; 之后依次对其他节点进行同样的操作. 相比于梯度下降, 这种方法实现比较容易, 比如在此题中, 梯度根本无法求得; 并且在某些特殊情形下, 坐标下降的收敛速度更快.

为了平衡模型经验误差和泛化误差, 采用交叉验证的方法来最优化节点位置. 具体而言, 就是将完整的数据集平均分成若干个子集, 每次选取其中一个子集作为验证数据集, 剩余的作为训练数据集; 在训练集上训练得到模型, 利用该模型预测验证集的输出, 并与验证集真实数据计算RMSE; 最后将所有RMSE求均值作为这组参数下模型的评价指标.

运行“利用CV对样条调参.py”, 可以得到四个最优节点位置, 并计算出优化后模型的RMSE:

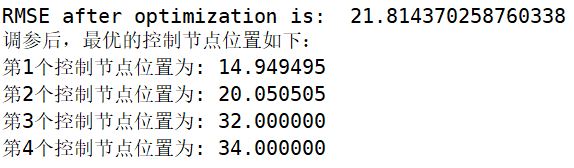


图9 三阶样条调参结果

可见相比调参前, 模型RMSE降低了很多, 而且也不用担心它的泛化能力, 因为这个最优模型是通过交叉验证选取出来的. 再来画出拟合曲线观察一下:

图表, 散点图

描述已自动生成

图10 调参后三阶样条拟合曲线

整条曲线在大约的区域, 拟合效果很好; 但在左侧边界位置, 则出现了一定的偏差, 而且该处的曲线形状是三次函数. 说明即使调参后, 样条在边界处拟合和泛化能力差的特点仍然存在. 为了缓解这一偏差, 可以考虑使用之前提到的自然三阶样条, 即在两个边界区域使用线性函数来拟合. 自然三阶样条的拟合曲线如图:

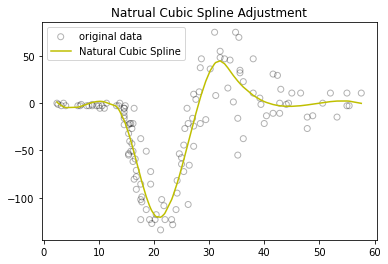


图11 使用自然样条进行调整

可见在自然样条的线性约束下, 曲线在边界的表现更好, 而且在内部仍然具有与原先一样的拟合效果. 因此选择将其作为本案例的最终回归结果.

2. 医疗费用预测(教材课后习题)

保险公司通常需要募集比花费在受益者的医疗服务上更多的年度保费. 因此, 精确预测医疗费用对保险公司具有重要价值. 本案例提供的数据集是从美国人口普查局的人口统计资料整理得出. 数据集共有1338个样本, 包含7个特征. 特征的具体信息如下表所示.

表1 医疗费用数据集特征

|  |  |
| --- | --- |
| 特征名称 | 特征说明 |
| age | 受益者年龄 |
| sex | 保单持有人性别 |
| bmi | 身体健康指数 |
| children | 保险计划中所包含的孩子/受抚养者的数量 |
| smoke | 被保险人是否吸烟 |
| region | 受益人的居住地 |
| charges | 已结算的医疗费用 |

请将“charges”作为目标特征, 构建回归模型并分析拟合的结果, 预测受益者的平均医疗费用. 数据集可在网络上自行下载.

## 案例二:

案例二是一个多元回归模型. 在案例二的实践中, 除了使用一些多元回归分析的基本方法, 还会使用之前介绍的核平滑、样条进行回归. 具体实践步骤如下:

## 步骤一: 读入数据集并进行一些准备工作

读入insurance.csv中的数据后用data.head(10)和data.describe()来获得数据集的全貌:

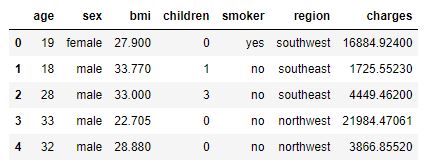
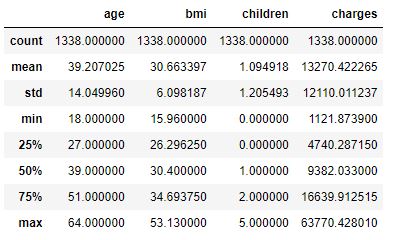


图12 insurance.csv数据集描述

可见数据共包含1338个样本, 7个特征, 无缺失值、离群值. 本案例是以“charges”作为目标特征, 进行多元回归. 剩余的6个特征中, “sex”“smoker”“region”是分类类型的特征, 因此需要进行onehot编码使其能够参与数值运算, 同时保持特征取值之间的距离不变.

另外, 考虑到本案例只给了一个数据文件, 因此对整个大样本进行划分, 取25%作为测试集, 剩下的作为训练集, 以验证回归模型的泛化效果.

步骤一的代码如下:

data = pd.read\_csv("C:\\Users\\16548\\Desktop\\导论作业\\T2\\insurance.csv")

y = data["charges"]

x = data.drop(labels="charges", axis=1)

x = pd.get\_dummies(x, columns=["sex", "smoker", "region"])

cols = ["sex\_female", "smoker\_no", "region\_northeast"]

x = x.drop(labels=cols, axis=1)

# 读入数据, 将charges作为目标特征, 剩余特征作为解释变量, 并对分类特征进行onehot编码

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

    x, y, test\_size=0.25, random\_state=2022

)

# 将完整数据集按1: 3的比例分割为测试集和训练集

## 步骤二: 使用核平滑进行回归

首先定义核平滑函数:

def KernelSmoothing(

    x\_train, y\_train, x\_test, h

):  # 核平滑, x\_train、y\_train是训练集, x\_test是测试集, h是窗宽

    n, m = x\_train.shape  # n是训练集样本容量, m是特征数量

    N = x\_test.shape[0]  # N是测试集样本容量

    x\_train = np.array(x\_train)

    y\_train = np.array(y\_train)

    x\_test = np.array(x\_test)

    yr = []  # yr记录测试集的预测结果

    for i in range(N):

        local\_index = []  # local\_index寻找与当前测试样本邻近的训练样本

        for j in range(n):

            if (

                np.linalg.norm((x\_train[j] - x\_test[i]).reshape(m, 1), ord=1, axis=0)

                <= h

            ):

                local\_index.append(j)

        Weight = np.exp(

            -(np.linalg.norm(x\_train - x\_test[i], ord=1, axis=1) \*\* 2) / (2 \* h\*\*2)

        ) / (

            np.sqrt(2 \* np.pi) \* h

        )  # 权函数采用高斯核函数

        Weight = Weight[local\_index]

        y\_hat = float(

            np.dot(Weight.reshape(1, len(Weight)), y\_train[local\_index])

            / np.sum(Weight)

        )

        yr.append(y\_hat)

return yr

之前已经介绍过核平滑的估计公式为: , 即利用预测点周围的训练样本的charges值对预测点进行估计, 至于“周围”这一概念, 则是由窗宽决定. 需要说明的是, 在高维的核平滑中, 两个样本点之间的距离由特征向量之差的范数给出; 这里则选用1-范数进行度量. 但是在这一题中, 核平滑的回归效果并不很好. 当选用窗宽分别为2, 4, 8时, 测试集样本点的残差分布及相应的RMSE如图:

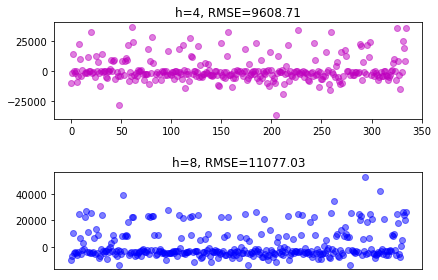


图13 窗宽为4和8时回归效果的对比

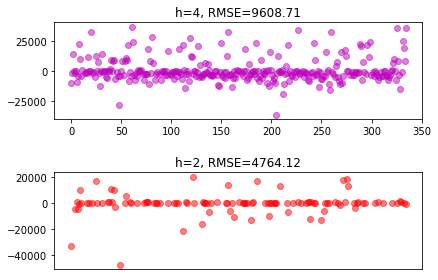


图14 窗宽为4和2时回归效果的对比

上面两幅残差分布图中, 纵轴表示, 可以用样本点距离直线的远近来度量残差的大小. 如之前所述, 小窗宽可以明显减少RMSE; 但是在上图中可以看到当减小时, 样本点的数量也在减小, 减小到时样本点已经明显少于的情形; 而样本点数量减少则意味着预测值有缺失. 这是因为当窗宽减小到一定程度时, “邻近”的要求也越来越严格, 即“邻近”区间变得很窄; 这会导致某些极端预测点没有“邻近”的训练集中的点, 也就无法对这些预测点进行预测.

## 步骤三: 使用最小二乘回归拟合模型

核平滑仅仅使用周围样本的信息对当前样本进行估计, 显示出了难以平衡经验误差和泛化误差的劣势. 步骤三考虑使用之前提到的最小二乘回归. 在建立回归模型时, 首先需要对真实模型的函数形式做出假设. 这就要求选择与响应变量charges相关性大的特征.

首先分析数据各特征之间（分类特征已经经过onehot编码）的相关系数矩阵（相关系数由进行估计）:

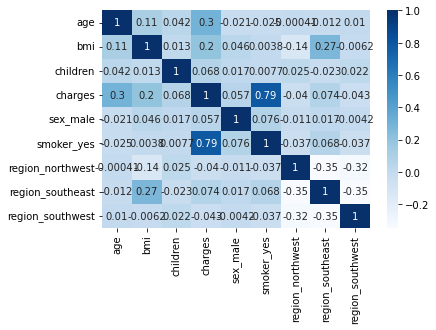


图15 特征相关系数矩阵

可以看出, 在热力图中, 目标特征charges与age、bmi、smoker\_yes的线性相关性比较强, 而与其他特征的线性关系较弱. 基于此, 接下来将建立两个回归模型: 一是使用全部特征, 并根据实际情况人为添加一些特征, 同时为了减少那些与charges相关性不大的特征对charges的影响, 防止过拟合, 使用岭回归对这些项的参数进行惩罚; 二是只使用age、bmi、smoker\_yes这三个特征, 直接对它们的参数进行最小二乘估计.

对于第一种方法, 先探索一些可能有利于解释模型的特征. 因age与charges相关性较高, 可以考虑添加新特征age2, 它的取值是age的平方, 从而扩大年龄对charges的影响; 其次, bmi可以作为一个阈值指标, 即超过某个值才会产生累积作用, 考虑新添加一个fat\_yes特征, 当bmi超过30时取值为1否则为0; 另外, 根据实际, 既肥胖又抽烟的人群所花费的保险金额平均而言必然多于其他人, 考虑添加一个特征fat\_smoker, 它是fat\_yes和smoker\_yes的交互项, 只有当它们取值均为1时fat\_smoker才取为1.

添加上述特征后, 使用5折交叉验证对惩罚项的系数进行选择, 回归的结果如下:

图表, 散点图

描述已自动生成

图16 岭回归模型下的残差分布

最终最佳的罚参数值为0.43, RMSE约为4208. 上图是回归残差分布图, 可见绝大多数点都落在0附近, 只有少量的点离0较远.

再来考虑第二种方法, 即使用普通的多元线性回归. 在训练集上训练模型后用于测试集, 算得模型的RMSE为6468, 与岭回归的结果相差不大, 说明岭回归的罚函数起到了一定的特征选择作用, 且新加入的特征应该有利于解释模型. 第二种方法在测试集上的残差分布图如下:

图表, 散点图

描述已自动生成

图17 只使用3个特征进行线性回归, 在测试集上的残差分布

同样, 大量的点均匀落在0两侧, 模型性能较为理想.

## 步骤四: 使用多元样条进行回归

在该步骤中将使用python中的pyearth库, 其提供了多元自适应样条回归算法. 它会自动地在多个特征中寻找非线性关系, 构建出最适合传入数据的样条模型. 它的工作由以下两步组成: 首先是前向传递步, 将会在幂样条中搜索最佳匹配项, 使训练集的MSE局部最小化; 接下来是剪枝步, 即在训练集中选择一个子集, 根据广义交叉验证分数来调整模型复杂度的惩罚项. 所谓广义交叉验证分数, 即, 式中为要调节的惩罚系数. 算法最终的结果是产生一系列分段函数, 它们在原始的样本空间中是非线性的.

MARS=Earth(smooth=False)

#构建多元自适应样条模型, 设定smooth参数为True, 使得一阶偏导数连续

MARS.fit(x\_train,y\_train)

#训练模型

y\_hat=MARS.predict(x\_test)

#y\_hat用于记录估计结果

执行以上三行代码, 即可实现对测试集样本charges值的估计. 之后根据y\_hat和y\_test, 计算回归的评价指标RMSE为4256, 决定系数为0.89. 这说明有89%的变差可以由该模型来解释, 回归效果较好. 以下是多元自适应样条模型的残差分布图:

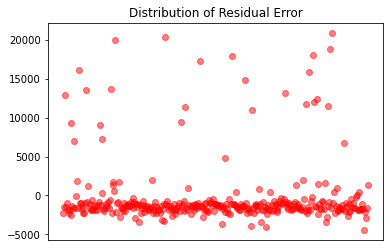


图18 多元自适应样条回归残差分布

综合以上所有实践步骤可以发现, 无论是非参数回归还是参数回归的方法, 都能对本案例完成较好的回归.