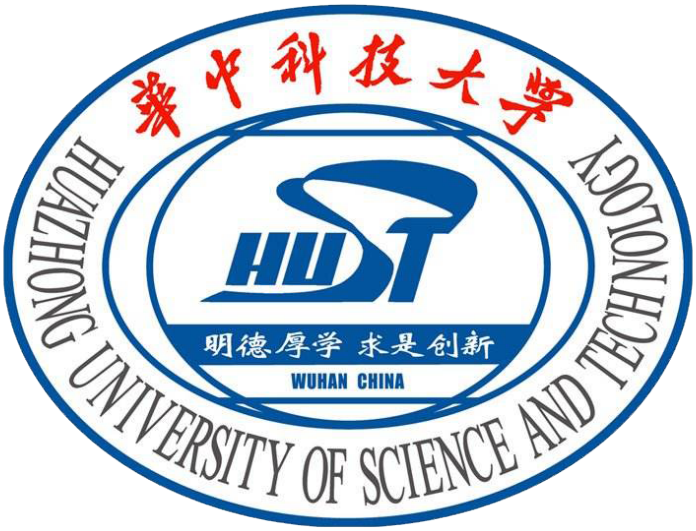
华中科技大学计算机科学与技术学院

《机器学习》课堂三结课报告



专　　业	计算机科学与技术
班　　级	ACM1901
学　　号	U201915010
姓　　名	李恒庄
成　　绩	
指导教师	何琨
时　　间	2021 年 12 月 9 日

目录

1 实验题目 基于差分进化算法的多段线性回归

1.1 摘要

本项目将多段线性回归问题分解了为三层子问题：已知断点位置的多段线性回归，已知断点数目的多段线性回归，未知断点数目的多段线性回归。同归递归的解决子问题，最终解决整个问题。本项目中涉及到比较重要的思想包括将多段线性回归归约为单个多维线性回归问题，并使用最小二乘求解器进行求解。同时，使用了差分进化算法优化残差平方和来得到全局最优断点位置。并且，在第三部分还提出了通过动态规划思想找到全局最优断点数的方法。系统也实现了本文提到的诸多思想，并做了很多利于使用者的优化，如果读者感兴趣，欢迎取用代码作进一步研究。

1.2 多段线行回归问题介绍

选择这个题目的源起是上了何琨老师的一节算法课后，何老师给我们留了一道作业题：考虑一个如图所示的数据集，我们可以观察到可以用多段线性函数来拟合这个数据集，我们需要确定将该数据集分为多少个聚类，并找到相应断点 ($breakpoint$) 使得代价函数 J（这里的代价函数可以有多种定义的方法）最小:

$$J(\boldsymbol{a}, \boldsymbol{b}, k) = \frac{1}{N} \sum_{i=1}^{k} \sum_{j=1}^{n_i} (a_i \cdot x_j^{(i)} + b_i - y_j^{(i)})^2$$

这道题乍看上去似乎很简单，不就是分段线性回归吗？但是当我课下实际下手做，发现事情并没有我想象的那么简单。粗略地查找相关文献之后，我发现断点数以及断点位置的确定仍然是解决这一问题的难点，很多学者都给出了解决办法，但是很多都是基于一定的数据特征以及实际应用限制，很少有给出一种普适而简约算法的。

由此，我发现研究此道题目对本科水平同学是一种有益的探索。所以我决定以这道题目作为我的机器学习课程的研究对象。

事实上，多段线性回归问题在近十几年都是一个比较活跃的问题，原因在于其所得拟合函数非常简单，对于一些轻量化或者定性问题的研究中都是时分有益的；在实际工业生产中，譬如滤波器设计等，面对繁杂的代价函数，人们更希望得到的函数是简单的；面对潜在信息未知的问题，多段线行回归能够在快速地给出对研究对象的粗略解释，方便人们掌握其中规律。

1.3 差分进化算法

在不断地检索文献后，我了解了一个经典算法——差分进化算法 [1](Differential Evolution Algorithm, DE) 能够用于解决这道问题中断点应该设置在哪里的问题（当得到断点后，问题给就变得非常简单了）。所以我决定以差分进化算法作为主体探索解决

多段线行回归问题的方法。差分进化算法在上世纪由 Storn 等人提出，接下来便被证明是收敛最快的进化算法，在接后的二十多年里不断被改进。

1.4 研究现状

多段线性回归早在本世纪初就已经在实际应用中被研究，从最开始的一维变量线性回归，经过无数学者的发展和研究，逐步扩展到多维回归问题以及多段非线性回归问题。相关研究从局限于小规模问题到大规模问题，从串行计算到并行分布式计算。同时，多段线性回归还被用来构建专家系统 [2]，是这一传统领域又有了新的突破；多段线行回归方法的引入可以使专家系统更加有效的获取所需信息，譬如可以使得医学中肿瘤评估的工作效率得到极大提高 [3]，通过多段线行回归改进的专家系统给出复杂医学归纳问题的答案。

限于篇幅，研究现状不再赘述。同时，本实验报告的多个部分均限于篇幅无法给出详细的证明过程、思维过程等，仅记载我大致的摸索过程以及遇到的相关问题。

2 实验要求

1. 自行从多种数据分布中采样构建不同数据集，包括线性决策边界和各种非线性决策边界；
2. 自行划分训练集与测试集；
3. 使用课程中学习过的分类器进行分类预测，包括核方法实现；
4. 评估不同模型在不同数据集上的表现，并给出相应分析及思考；
5. 自主拓展探索;
6. 严禁直接调用已经封装好的各类机器学习库（包括但不限于 sklearn），但可以用 NumPy 等数学运算库，严禁抄袭网上代码。

3 算法设计

综合来讲，我们需要解决的问题分为三个步骤：

- 确定断点 ($breakpoints$) 的数目；
- 确定各断点的位置（已知第一个断点和最后一个断点分别为 x_1 和 x_n）；
- 通过断点对数据集进行分段线性规划。

根据我之前的分析，这三个子问题中，第一个子问题难度最大，学界至今没有得出一个令所有人都信服的算法；而对于第二个子问题，根据先前的描述，可以通过使用差分进化算法来找到全局最优的断点位置，进而将问题归约为第三个子问题；第三个子问题最为简单，仅需将数据集划分为多段，并在多段上面使用最小二乘法即可。

根据子问题难度，接下来行文依据子问题倒序进行，方便读者渐进了解多段线性回归的解题思想。

3.1 已知断点位置的多段线性回归

首先，我们要明确，我们需要找到一条连续的分段线性函数。在实际应用中，出现情况最多的也是连续的分段线性函数；同时，设定这个要求也有我的一个私心，不连续的分段线性函数对我来说不够优雅（LOL）。假设我们有一个大小为 n 一维数据集，其中 x 为自变量，y 为因变量。我们的可以将数据集表示为以下这种形式：

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

同时，为了后续讨论方便，这里假设数据集已经根据 x_i 排好序了。在数据集中，我们假设 $x_1 < x_2 < x_3 < x_4 < \cdots < x_n$。注意没有 $x_i = x_j$ 的情况，这也是为了后续讨论方便。因为如果两数据自变量相等，且恰好断点在该处，则在回归过程中无法得到一条连续的多段线性函数。但是真实数据集中无法避免出现自变量相等的情况，那么我们有以下解决办法：

- 对于两个自变量相等的点，随机删去其中一个。当然，这可能也会以引发一些其他的问题，比如使得数据集出现类别不平衡 ($class - imbalance$) 的问题；
- 对于自变量相等的点，我们将其归约为一个数据点，数据点的因变量为这些点因变量的平均值，即 $y' = \frac{1}{m} \cdot \sum_{i=1}^{m} y_i$，这样做也会造成问题，如果实际数据集对应

的斜率很大或实际数据集本身就是分段的线性回归，则这种操作相当于人为的将数据集强制转换成了连续的线性分段函数；

· 将自变量相等的点的自变量加上一个随机的偏移量，强行使得所有的点的自变量都不相同，这种是最无脑的。

以上问题解决办法还有很多，实际问题中根据需求具体选择即可。

接下来继续讨论该子问题。当我们将数据集根据自变量 x 排好序，且已知断点位置的情况下，我们最终解的形式应该如下所示 [4]：

$$\mathbf{y}(x)=\begin{cases}\eta_1+\beta_1(x-b_1) & b_1<x\le b_2\\ \eta_2+\beta_2(x-b_2) & b_2<x\le b_3\\ \vdots\quad\vdots & \\ \eta_n+\beta_{n_b}(x-b_{n_b-1}) & b_{n-1}<x\le b_{n_b}\end{cases}$$

其中，断点为 $b_1>b_2<\cdots>b_{n_b}$，总共有 n_b 个断点，且已经排好序。先前已经提到，其中 $b_1=x_1$，且 $b_{n_b}=x_n$。那么问题已经可以得到很好的解决了，只需对每一个分段使用最小二乘法即可。接下来介绍一种处理方法，可以只通过一次最小二乘法即可得到结果。

首先，我们对解的形式进行以下变换：

$$\mathbf{y}(x)=\begin{cases}\beta_1+\beta_2(x-b_1) & b_1\le x\le b_2\\ \beta_1+\beta_2(x-b_1)+\beta_3(x-b_2) & b_2<x\le b_3\\ \vdots\quad\vdots & \\ \beta_1+\beta_2(x-b_1)+\beta_3(x-b_2)+\cdots+\beta_{n_b+1}(x-b_{n_b-1}) & b_{n-1}<x\le b_{n_b}\end{cases}$$

为什么可以进行以上变换呢，因为我们考虑的是连续的线性回归问题，对于每一段线性函数，我们都可以看作其是在前一个线性函数的基础上加上一个线性函数（该线性函数的系数可以是负数)。例如,对于式中的第二段线性函数 $y(x)=\beta_1+\beta_2(x-b_1)+\beta_3(x-b_2)\quad b_2<x\le b_3$, 我们可以看作其在第一段线性函数 $y(x)=\beta_1+\beta_2(x-b_1)\quad b_1\le x\le b_2$ 的基础上加上一个 $y(x)=\beta_3(x-b_2)b_2\le x\le b_3$ 的线性函数。依此类推，得到解的另一种形式。

进一步，我们可以将该解写成矩阵乘的形式，如下所示：

$$\begin{bmatrix}1 & x_1-b_1 & (x_1-b_2)s_{x_1>b_2} & (x_1-b_3)s_{x_1>b_3} & \cdots & (x_1-b_{n_b-1})s_{x_1>b_{n_b-1}}\\ 1 & x_2-b_1 & (x_2-b_2)s_{x_2>b_2} & (x_2-b_3)s_{x_2>b_3} & \cdots & (x_2-b_{n_b-1})s_{x_2>b_{n_b-1}}\\ & & \vdots\quad\vdots\quad\vdots\quad\vdots\quad\vdots\quad\vdots & & & \\ 1 & x_n-b_1 & (x_n-b_2)s_{x_n>b_2} & (x_n-b_3)s_{x_n>b_3} & \cdots & (x_n-b_{n_b-1})s_{x_n>b_{n_b-1}}\end{bmatrix}\begin{bmatrix}\beta_1\\ \beta_2\\ \vdots\\ \beta_{n_b}\end{bmatrix}=\begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_n\end{bmatrix}$$

其中，$s_{x_i>b_j}$ 函数定义为以下的形式：

$$s_{x_i>b_j} = \begin{cases} 0 & x \le b_j \\ 1 & x > b_j \end{cases}$$

至此，已知断点位置的多段线行回归问题即将解决。通过函数 $s_{x_i>b_j}$ 的转换，式中的回归矩阵可以被转换为下三角矩阵，利于后续的矩阵计算。而求解该问题可以使用 $NumPy$ 中的最小二乘求解器 $numpy \quad lstsq$ 完成矩阵的运算 [5]。

上述等式可以表示为以下形式 [6]：

$$\mathbf{A}\beta = \mathbf{y}$$

因此，解的形式为：

$$\beta = (\mathbf{A^T A})^{-1}\mathbf{A^T y}$$

因此数据集的残差可以表示为：

$$\mathbf{e} = \mathbf{A}\beta - \mathbf{y}$$

其中，$\mathbf{e}$ 为 n 维向量，因此残差平方和 $(residual\ sum\ of\ squares)$ 为：

$$SS_{res} = \mathbf{e^T e}$$

接下来根据多段线性回归的特点，简要介绍和修正一些用于评估模型的参数。首先是因变量总平方和为：

$$SS_{tot} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

因此，判定系数为：

$$\mathbf{R^2} = 1 - \frac{SS_{res}}{SS_{tot}}$$

已知样本量 n 和断点个数 n_b，残差平方和的无偏估计为：

$$\hat{\sigma}^2 = \frac{SS_{res}}{n - n_b}$$

因此，假设数服从正态分布，则对于每个 β_i，其中 $1 \le i \le n_b$，其标准差为：

$$SE(\beta_i) = \sqrt{\hat{\sigma}^2 [A^T A]^{-1}_{ii}}$$

3.2 已知断点数目的多段线性回归

接下来介绍已知断点数目的多段线性回归。根据已知断点位置的多段线性回归的介绍，如果我们能够在合理时间复杂度内找到最佳位置，就可以将已知断点数目的多段线性回归归约到已知断点位置的多段线性回归问题。首先，我们给出已知断点数目的多段线性回归问题的具体描述：

$$given\ b_i, 1 \le i \le n_b, x_1 \le b_i \le x_n (b_1 = x_1, b_{n_b} = x_n)$$
$$minimize\ SS_{res}(\mathbf{b}), \mathbf{b} = [b_2, b_3, \cdots, b_{n_b-1}]^T$$

接下来介绍找出全局最优解的算法——差分进化算法 [1]（Differential Evolution Algorithm）。差分进化算法是 Rainer Storn 和 Kenneth Price 在 1997 年提出的启发式算法，后续在各种竞赛中都被证明是目前收敛最快的进化算法。在随后的 20 多年中，差分进化算法得到了不断地优化，并衍生出多种变体。本次实验中仅采用最经典的差分进化算法。接下来简要介绍差分进化算法流程。

3.2.1 创建种群

首先需要设置种群的初始数量，一般设置为 10 到 20，设为 $popsize(population\ size)$。随后，生成 $popsize$ 个 $(0, 1)$ 的 d 维随机数（其中 d 为自变量的维数），然后再将将随机数映射到自变量的域。这样我们就得到了 $popsize$ 个初始种群。接下来使用需要被优化的函数来评价这个种群，即使用该函数计算得到每个个体的值。得到每个个体的值后，我们就可以从中找出我们最需要的那个个体（本实验中，因为要最小化，所以我们找出具有最小值的个体即可）。

3.2.2 种群突变

在现有种群中找到最优的那个个体之后，我们就可以进行突变 $(mutation)$ 了。在剩余的种群里选择三个个体作为突变源，不妨记作 a, b, c。突变的核心操作是，根据 b, c 的差异来改变 a 的值，即将 b, c 的差异乘以突变因子（一般为 $[0.5, 2.0]$，突变因子过大或过小可能会减慢收敛速度），再加上 a 的原始数据就得到了一个新的个体。突变后的新个体的各维变量可能超过自变量的域，所以还要将新值归约到自变量的域中。

3.2.3 种群重组

接下来就是用新的个体中的数据替换当前最优个体中的某些数据，以期达到进化的目的。对于每一维变量，其重组的概率均为一个值，这个值一般设定为 0.7，可根据变量维数大小做相应调整。最终的重组结果也遵循二项式分布。

3.2.4 种群替代

得到重组个体之后，我们就可以使用待优化函数来对新个体进行评估。如果该新个体的评价好于当前最优个体，就用该新个体替换它。当然，对于整个种群中的个体，只要新个体的评价高于该个体（本实验中，函数值更小），都会被替换。

下面给出差分进化算法的伪代码：

Algorithm 1: Differential Evolution

Data: Population: M, Dimension D, Generation T

Result: The best vector (solution) Δ

$t \leftarrow 1(initialization)$;
for $i = 1$ *to* M **do**
    for $j = 1$ *to* D **do**
        $x_{i,t}^j = x_{min}^j + rand(0,1) \cdot (x_{max}^j - x_{min}^j)$;
while $(|f(\Delta)| \geq \varepsilon)$ *or* $(t \leq T)$ **do**
    for $i = 1$ *to* M **do**
        $\triangleright(Mutation\ and\ Crossover)$;
        for $j = 1$ *to* D **do**
            $v_{i,t}^j = Mutation(x_{i,t}^j)$;
            $u_{i,t}^j = Crossover(x_{i,t}^j, v_{i,t}^j)$;
        $\triangleright(Greedy\ Selection)$;
        if $f(\mathbf{u}_{i,t}) < f(\mathbf{x}_{i,t})$ **then**
            $\mathbf{x}_{i,t} \leftarrow \mathbf{u}_{i,t}$;
            if $f(\mathbf{x}_{i,t}) < f(\Delta)$ **then**
                $\Delta \leftarrow \mathbf{x}_{i,j}$;
        else
            $\mathbf{x}_{i,t} \leftarrow \mathbf{x}_{i,t}$;
    $t \leftarrow t + 1$;
Return *the best vector* Δ

实现上述差分进化算法之后，我们就可以将 $SS_{res}(\mathbf{b})$ 作为待优化的评价函数传入算法，并最终得到全局最优的 $\mathbf{b}$。这里要提的是，差分进化算法找到全局最优解的开销随着维数的增加而指数性增加。也就是说，当我们断点数目增加，求解已知断点数目的分段线性回归问题的开销将呈指数增加。这一特性我们在接下的一节未知断点数目的多段线性回归还会提到。

3.3 未知断点数目的多段线性回归

开宗明义，在题目给定的条件下，求解未知断点数的多段线性回归目前只有非常少的文献提及。且在实际查阅文献过程中，我也发现许多学者得到全局最优的断点数时，均进行了一定的假定或者对数据进行了一定的预处理。其中有通过将因变量和自变量组合成为 $d+1$ 维的数据，将问题归约为面聚类问题再进行求解的 [7]；也有只识别单个输入特征，并在该特征上将样本分成互补区域，对每个区域局部拟合一个不同的线性回归函数的解决方案 [8]。总的来说，No Free Lunch 定理提出，充分利用先验信息是提升学习性能的最有效途径之一；进一步来说，在本问题中，如果无法利用所有数据集的特征，我们将无法找出全局最优的断点数目。

因为本人能力有限，暂时还未深刻理解这些方法的内涵。在此，我提出一种基于动态规划思想的可能可行的算法。这也是本学期算法课堂上同学首先贡献出来的思想精华，我只是进行一些修正和补充。

对于数据集中的一个子集 $A_{\mathbf{x}_i,\cdots,\mathbf{x}_{i+j}} = (\mathbf{X}, \mathbf{Y})$，借鉴前面3.1的推导，我们可根据残差平方和度量子序列的损失：

$$SS_{res}(A_{\mathbf{x}_i,\cdots,\mathbf{x}_{i+j}}) = \mathbf{e}^{\mathbf{T}}\mathbf{e}$$

接下来用 $f(i)$ 表示前 i 项数据分成若干段得到的最小损失（$f(0)$ 初始化为 0），则动态规划的转移方程为：

$$f(i) = \min_{0 \le j < i}(f(j) + SS_{res}(A_{\mathbf{x}_{j+1},\cdots,\mathbf{x}_i}) + c) \quad (1 \le i \le n)$$

即，对于第 i 个数据点，考虑其分别和前面若干个结点组成一段线性函数（从 1 到 $i-1$），再加上因为多处分段而出现的惩罚系数 c。因为计算每个 $f(i)$ 都需要遍历 $f(j), j < i$，所以该动态规划算法的时间复杂是 $O(n^2)$。对于每个 $f(i)$ 的计算都可以记录对应的最优分段数。当算法运行完后，$f(n)$ 对应的分段数就是全局最优的分段数/。

限于篇幅，这里就不再证明该算法的正确性。当然，最终实现的过程中，没有实现该算法，因为在3.1中已经将问题归约为单个多元线性回归问题，如果采用该算法来计算，则会增加额外的开销。因此，在最终实现的版本中，为了方便系统实现，我采用遍历枚举的方法取得局部最优的断点数。同时，该动态规划算法还需要引进惩罚系数 c。确定惩罚系数本身的工作量也十分巨大，因此在最终实现的时候没有采用该算法。

在3.2中也提到，在增加分段数的过程中，差分进化算法的开销将指数级增加。所以，我将分段数限制在一个合理范围内，而实际上分段数的所有取值有 n 中（考虑每一个数据点作为一个分段）。

4 实验环境与平台

4.1 环境

开发机器配置
Processor Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
Installed RAM 8.00 GB (7.81 GB usable)
Device ID 883D3DDC-2BC4-44EE-AEFD-6BFE3663606D
Product ID 00330-80000-00000-AA248
System type 64-bit operating system, x64-based processor
开发系统
Edition Windows 10 Pro
Version 20H2
Installed on 2020-12-07
OS build 19042.1348
Experience Windows Feature Experience Pack 120.2212.3920.0

4.2 实验平台

PyCharm 2021.2.3 (Professional Edition)
Build PY-212.5457.59, built on October 19, 2021
Licensed to Hengzhuang Li
Subscription is active until December 26, 2023.
Evaluation purpose only.
Runtime version: 11.0.12+7-b1504.40 amd64
VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.
Windows 10 10.0
GC: G1 Young Generation, G1 Old Generation
Memory: 1000M
Cores: 12
python 版本
3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)]

5 程序实现

5.1 系统整体架构

本次实验的主要探究对象是多段线性回归问题。我将多段线性回归拆分成了三个子问题，且三个子问题是迭代求解的。因此在系统层面，我也迭代地对系统进行构建，使得功能逐层封装，有利于实验整体思想脉络的把握以及方便阅读者更快理解我的代码。我的设计是，首先实现一个 SimplePLR 类，用于对已知断点位置的多段线性回归问题进行建模和求解。接下来，在类 SimplePLR 的基础上，构建 PLR 类继承 SimplePLR 类，并实现已知断点数目的多段线性回归问题，包括求解全局最优的断点位置等。在求解全局最优的断点位置的时候需要使用差分进化算法，因此再实现一个 DifferentialEvolution 类，实现需要的差分进化算法的静态函数，包括不同版本：正式版和调试版。最后，在主程序中实现了以上三个类的多个测试函数以及最终的实际应用程序。类实现的 UML 图如 5.1 所示。

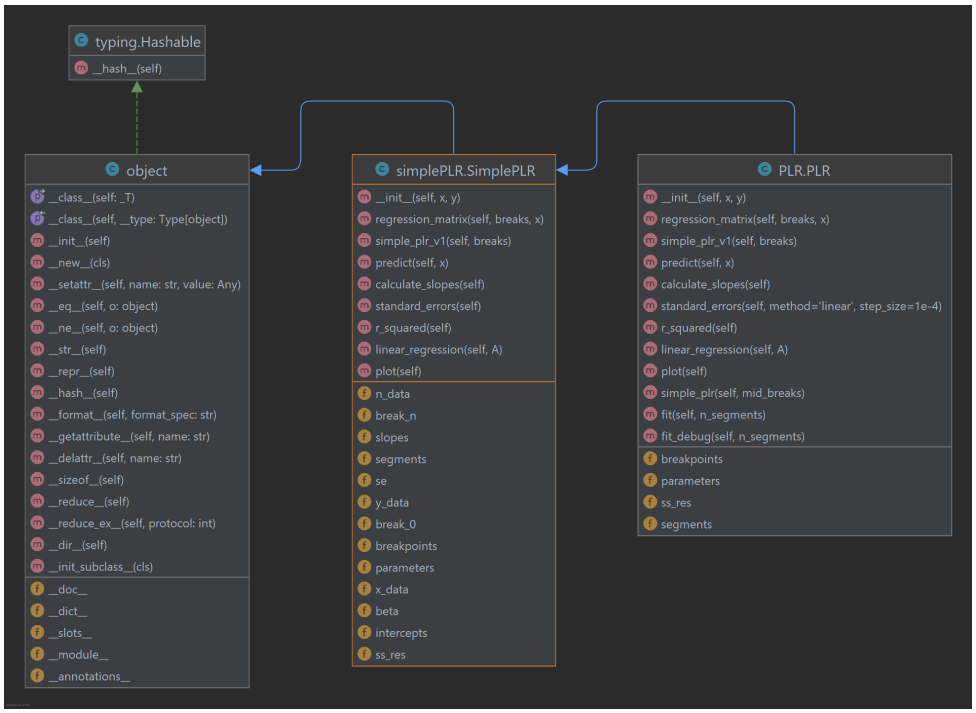


图 5.1: 系统 UML

5.2 SimplePLR 类实现

这个类中，首先需要输入数据作为模型的训练数据；在该子问题中，问题进一步细化为：生成回归矩阵，求解普通线性回归问题，计算各分段函数斜率截距，计算标准差，计算判定系数，对接下来的数据进行预测。这些子问题分别由函数 $regression\_matrix(breaka, x)$，$simple\_plr\_v1(breaks) \rightarrow linear\_regression(A), calculate\_slopes(), standard\_errors()$，

$r\_squared()$，$predict(x)$ 实现。

在进行建模过程中，需要先调用 $SimplePLR$ 创建模型对象，并向其中传入自变量和应变量数据。注意，这里的数据可以为 list 对象也可以为 np.ndarray 对象。随后，程序会初始化建立训练集，并初始化相关变量。在创建好模型后，就可以直接调用 $simple\_plr\_v1(breaks)$ 函数将数据集划分为多端的线性函数。需要注意的是，在解决已知断点位置的多段线行规划问题中，需要事先人工找出断点位置。读者如果感兴趣，可以继续读下去，稍后我会告诉你如何传入断点位置。随后，就可以进行多项操作，比如进行数据的预测 $predict(x)$，其中 x 为新的数据。需要注意的是，这里的测试数据的值域需要包含于训练集的值域。这样规定是合理，因为多段线性回归问题的本身决定了我们只能预测已知域中的数据，而对于已知域外的数据，模型只会利用第一段或者最后一段的线性函数进行预测，而这样显然会出现问题，因为我们没有从已知数据集推断后续数据集的能力，这样的问题交给其它课题。

同时，使用者还可以调用 $standard\_errors()$，$r\_squared()$ 函数来计算标准差和判定系数对训练结果进行一个初步的评估。同时，使用者也可以直接查看模型中的 ss_res 值，来得到残差平方和。需要注意的是，在后续的全局优化中，我正是使用残差平方和来评价一种断点位置的优劣的。

SimplePLR 类中各种函数调用依赖关系如下 5.2 图所示：

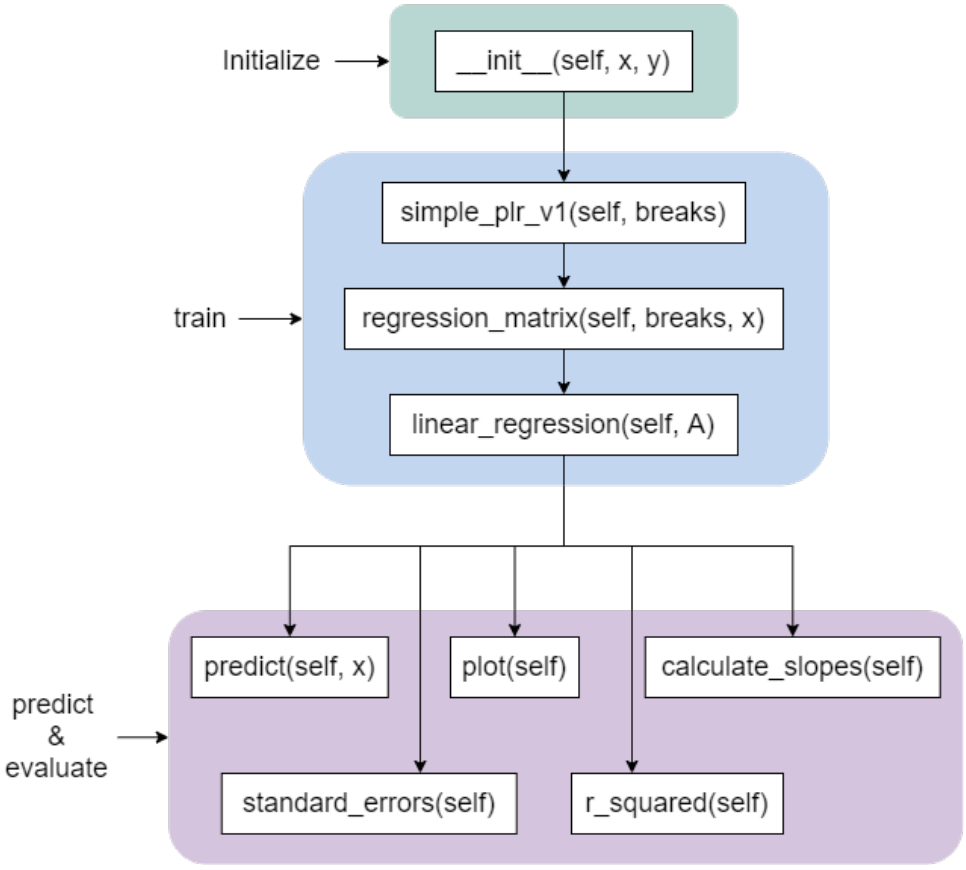


图 5.2: 系统 UML

其中，$regression\_matrix(breaka, x)$ 的计算方法完全依照 已知断点位置的多段线性回归 节给出的计算方法。$plot()$ 用于数据可视化，输出展示的图像上，有源数据和拟合多段线性函数，供使用者从整体上评估模型的优劣。

需要注意的是，由于数据集不一定规范，所以根据 已知断点位置的多段线性回归 节计算标准差时，可能出现回归矩阵奇异的情况，此时 $standard\_errors()$ 会抛出奇异矩阵的异常，此时使用者可以考虑修正数据集或者不再考虑标准差。因为本次实验时间紧迫，所以没有采用更复杂的处理方法。在实际完整的项目中，可以实现对数据进行预处理，使得数据集中不存在自变量完全相同的两个数据，从而避免回归矩阵奇异的情况。

5.3 PLR 类实现

PLR 类是在继承类 SimplePLR 类的基础上实现的。综合来讲，PLR 类完成的工作就是找到全局最优的断点位置，而断点中，已经有两个断点位置确定，第一个断点和最后一个断点。也就是说，我们需要确定的位置数为 $breakpoints - 2$。为此，重新实现一个函数 $simple\_plr(mid_breaks)$，每次只传入中间 $breakspoints - 2$ 个参数，用该函数的返回结果 ss_res 作为评估的指标，即需要优化断点位置以使得残差平方和最小。当然，如果读者还想引入其他的指标来优化断点位置，可以仿照 $simple\_plr(mid_breaks)$ 实现评估函数。

同时，为了方便调试得到中间结果和观察迭代过程中的变化，我还实现了一种调试版本的 $fit\_debug(n\_segments)$ 函数，其与普通的 $fit(n\_segments)$ 函数的唯一区别就是，其调用的是调试版本的差分进化算法，具体实现在 DifferentialEvolution 类实现 节中介绍。当需要中间调试信息的时候，使用者可以调用调试版本的 $fit\_debug(n\_segments)$ 函数即可，函数中已经将每次迭代后结果呈现在 plt 中，最终打印输出图像就可以观察到待优化函数随着迭代次数的增加而变化的曲线。

在 fit 函数中，首先根据给定的分段数和收尾断点位置来生成 $bounds$ 作为中间断点位置的域；然后调用 $differential\_evolution(simple\_plr, bounds)$ 函数来得到全局最优断点位置（其中 $simple\_plr$ 是传入的待优化函数变量）；得到断点位置后就可以调用从 SimplePLR 继承的 $simple\_plr\_v1$ 函数来解决已知断点位置的多段线性回归问题。

5.4 DifferentialEvolution 类实现

DifferentialEvolution 类封装了差分进化算法。在本次实验中，为了简便起见，差分进化算法采用 $rand/1/bin$ 突变策略。

接下来简要介绍差分进化算法的策略组成。其中 $rand$ 字段可选 $rand, best, rand - to - best$ 策略等，分别表示随机选取个体，选取最佳个体和随机选择个体以及最佳个体；1 字段可选 1 和 2，分别表示取一对两个体之差和两对两个体只差来生成突变个体，即

表 5.1: 差分进化算法策略选择

重组策略	方案
Rand/1	$x_{mut} = x_{r1} + F(x_{r2} - x_{r3})$
Rand/2	$x_{mut} = x_{r1} + F(x_{r2} - x_{r3} + x_{r4} - x_{r5})$
Best/1	$x_{best} = x_{r1} + F(x_{r2} - x_{r3})$
Best/2	$x_{best} = x_{r1} + F(x_{r2} - x_{r3} + x_{r4} - x_{r5})$
Rand-to-Best/1	$x_{mut} = x_{r1} + F_1(x_{r2} - x_{r3}) + F_2(x_{best} - x_{r1})$

差向量的数量；*bin* 字段可选 *bin* 和 *exp*，分别表示二项式模式和指数模式，二项式模式表示每个变量都具有相同的概率进行重组，指数模式表示随机选取序列中连续的长度为 n 的子序列进行重组。差分进化算法的策略可以通过排列组合上述字段形成多种策略，每种策略在各种不同的实际应用中都有不同的特性，使用者应该根据实际需求选取合适的进化策略。部分进化策略如表 5.1 所示。

具体实现在此就不再赘述，读者可以参考 已知断点数目的多段线性回归 中对差分进化算法的介绍，或者直接阅读源文件中的 DiferentialEvolution 类的源码，我已经写了较为详细的注释方便理解。

5.5 主程序实现

主程序 (*main.py*) 主要包含一些对代码模块功能测试的函数以及最后的一个小应用。接下来逐个介绍个函数的作用。

(1) simple_plr_tester1()

使用 SimplePLR 类构建模型，进行已知断点位置的多段线性回归。函数中已经给出了样例数据以及样例断点。需要注意的，断点还需要传入首位两个断点，即数据集中自变量的最小值和最大值。

多次改变断点的数目和位置，就可以得到不同的拟合曲线，对比各种不同曲线，即相应的标准差和残差平方和等，我们可以大致判断出那种断点位置是最佳的，不过也可以从中感受到手动设置断点位置的麻烦之处。

(2) plr_tester1()

使用 PLR 类构建模型，进行已知断点数目位置的多段线性回归。因为需要确定全局最优的断点位置，所以在构建模型之后还需要调用 fit 函数进行优化。

在这一步中，可以多次改变分段的数目来观察不同分段数目下模型的拟合效果，这可以通过观察可视化曲线以及相关指标来评价。

(3) plr_tester2()

此测试函数是在 $plr\_tester1()$ 的基础上构建的，通过遍历多个分段数，每个分段数都输出拟合曲线的图像，最终输出判定系数随着分段数的增加而变化的曲线。通过观察这些曲线，我们能够得到一些粗略的结论，并能够理解模型运行背后的动态变化规律。

(4) de_tester1()

这个测试程序是单独用来测试差分进化算法的收敛速度的。首先给出我们的实例函数 $lambdax : sum(x**2)/d$，其中 d 表示数据的维数，取值为 $8, 16, 32, 64$，x 为维数为 d 的自变量，变化范围为 $(-100, 100)$。

遍历四种维数，分别调用调式版本差分进化算法，得到每种维数下，随着迭代次数的增加实例函数的值的变化规律，从而观察差分进化算法收敛速度和维数之间的关系。

(5) de_tester2()

该测试函数是在 de_tester1() 的基础上进行的。对于给定的实例数据集，通过 PLR 类建立模型。给定一系列分段数，遍历这些分段数，并在每次遍历时，调用调试版本的 $fit()$ 函数，这样就得到了每个分段数对应的差分进化算法收敛情况与迭代次数的关系。将这些曲线绘制到同一张图中就可以直观的观察到不同分段数对收敛速度的影响，并判断出那种分段数相对来说最优。

(6) de_tester3()

此测试函数是为了观察分段线性回归模型对于非多项式函数的拟合情况。生成一个余弦函数的数据集，并在每个数据的因变量上加上随机的偏置量，来模拟自然情况下的分布。随后，设置不同的分段数，在该数据集上运行分段线性回归，将回归曲线绘制在同一张图上，观察不同曲线拟合的异同点。从这个测试中也可以看到，分段线性回归模型对于大致分析数据集分布和趋势的优势。该模型非常简单，并能直观反映出数据集的大致分布。

(7) application()

接下来是使用分段线性模型来进行一个简单的小应用。此函数是用于获取数据集的函数。这个小应用是对过去一段时间中上证指数每日走势的一个分段线性回归。做这个应用的目的不是为了预测未来走势，而是从走势图中更进一步抽象出在这段时间内上证指数的变化拐点（前面也提到过，用分段线性函数预测域外值是十分不明

智的做法)。为此，我选取 *baostock* 库作为我获取大盘走势数据的 API。此函数获取数据的流程也十分简单，首先需登录，然后请求数据，将获取的每列数据整合进 pd.DataFrame 中，最后将数据存储在项目文件夹下的

/data/history_A_stock_k_data.csv。

(8) app_plr()

这个函数用来从获取的数据取出我们所需要的部分，并根据该数据通过类 PLR 建立模型，进行数据的训练，并最终得到训练结果。设置不同的分段数目，得到了不同的分段函数，将这些函数绘制在一张图像中，这样我们就可以大致观察出不同分段数对模型的拟合情况的好坏。

需要注意的是，根据前文 未知断点数目的多段线性回归 的论述，在本次实验中仅通过在一定范围内遍历分段数目来确定效果较为优良的分段数，而没有实现动态规划思想。

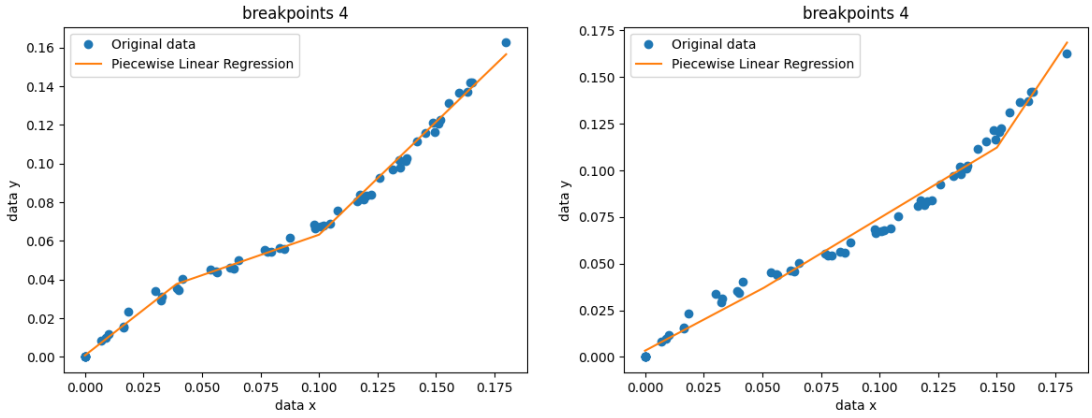
6 实验结果及分析

6.1 测试 1

使用 SimplePLR 类构建模型，进行已知断点位置的多段线性回归。对于给定的实例数据集，我们给出两种断点位置方案，分别为:

$$[min(x\_data), 0.039, 0.10, max(x\_data)]$$
$$[min(x\_data), 0.050, 0.15, max(x\_data)]$$

程序运行过后，所得到的结果分别为 1(a) 和 1(b):



(a) (b)

图 6.1: 已知断点位置多段线性回归

同时，上述两种断点位置的分配对应的标准差和判定系数分别为:

$$SE(\beta_1) = [0.00096346, 0.0388971, 0.05737514, 0.04040854] \ R_1^2 = 0.99610201$$
$$SE(\beta_2) = [0.00175845, 0.05263311, 0.07186705, 0.16834045] \ R_2^2 = 0.98563604$$

通过观察上述曲线图像和相关系数可以知道，第一个断点位置分配更优。从这个测试中，我们也可以知道标准差和判定系数可以用于判断分段线性回归的效果。

6.2 测试 2

使用 PLR 类构建模型，进行已知断点数目位置的多段线性回归。图 6.2 是分段数为 5 的运行结果，可以看到拟合曲线表现较不错。

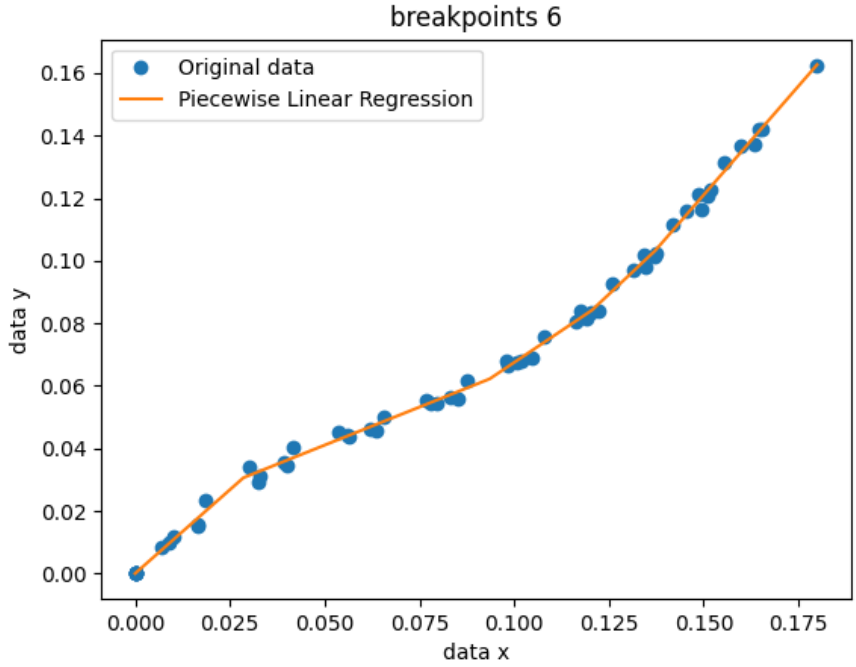


图 6.2: PLR 模型构建全局最优的断点位置

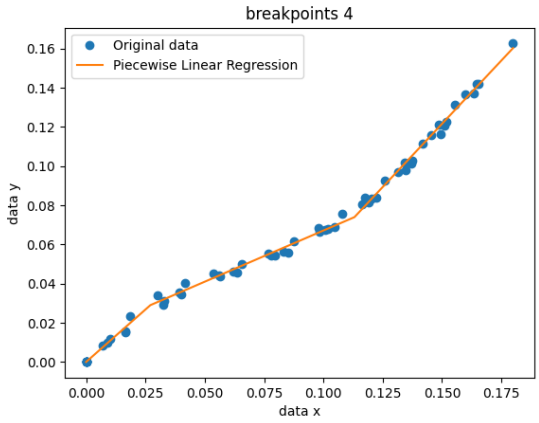
如此还可以继续改变分段的数目，并使用差分进化算法得到全局最佳的断点位置。这种思想在 各种不同分段数对应的线性拟合 中得到了体现。

6.3 测试 3

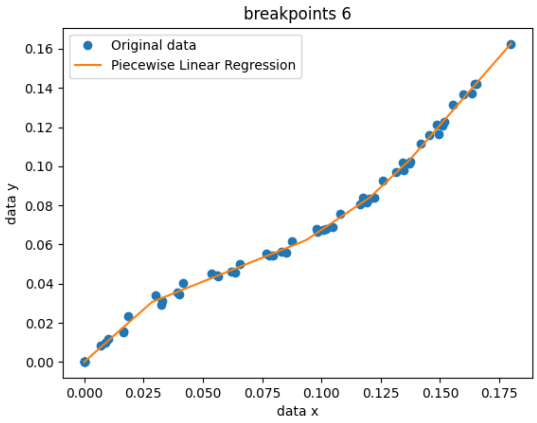
通过遍历多个分段数，每个分段数都输出拟合曲线的图像。同时生成判定系数随分段数增加而变化的曲线图。以下甄选出了一些典型分段数对应的拟合曲线，如图 6.3 所示。同时，还绘制了判定系数的变化曲线，如图 6.4 所示。

从图中可以看到，当分段数增加时，虽然标准差和判定系数都会变得更好，但是从图中直观来看，分段数增加到一定数目之后，拟合曲线就会出现毛刺（这一般是不被期望的），也就是说分段数过多会导致数据过拟合的情况，因此并不是分段数越多越优。

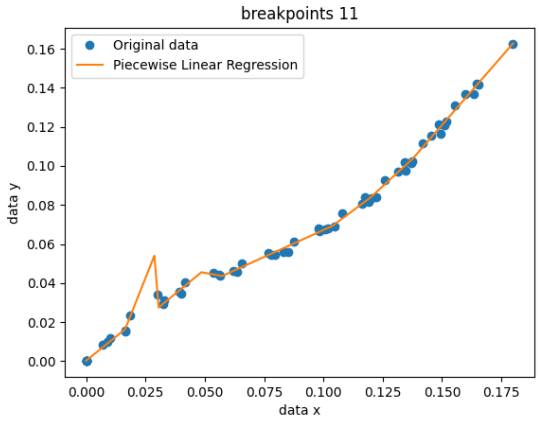
从判别系数随分段数变化曲线也可以看到，开始时，判别系数随分段数增加而显著增加；而当分段数增加到一定量后继续增加时，判别系数增加速度显著下降，并逐渐趋近 1。以上测试结果对我们实际应用中的启示是，应该选择合适的分段数使得各种判定的数值足够优秀且避免数据过拟合的情况。



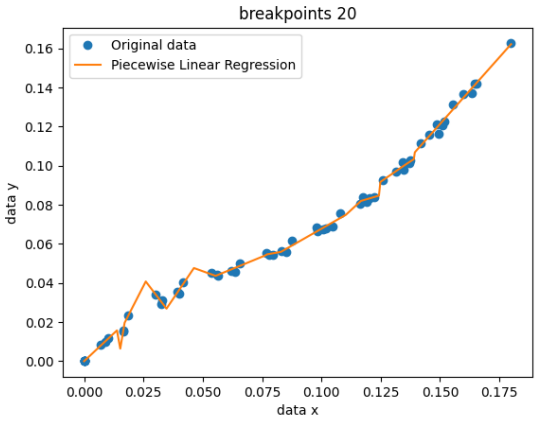
(a) segment 3



(b) segment 5



(c) segment 10



(d) segment 19

图 6.3: 各种不同分段数对应的线性拟合

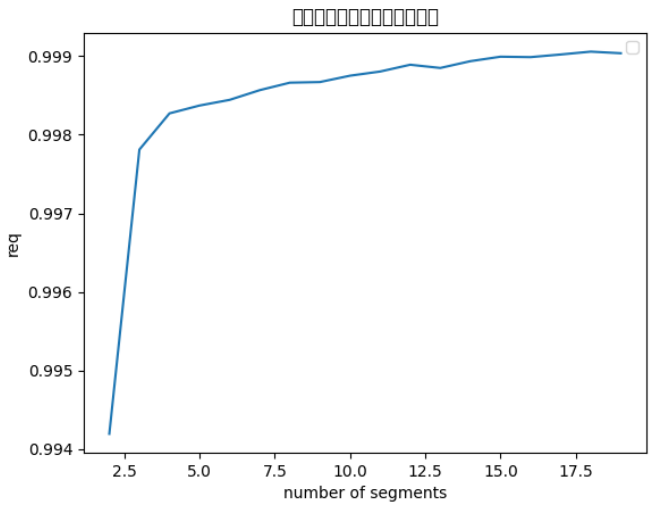


图 6.4: 判别系数随分段数增加的变化

6.4 测试 4

测试差分进化算法的收敛速度。这个测试对不同维数的数据，根据相同的评价函数，得到评价函数的最小值与迭代次数之间的关系如图 6.5 所示。

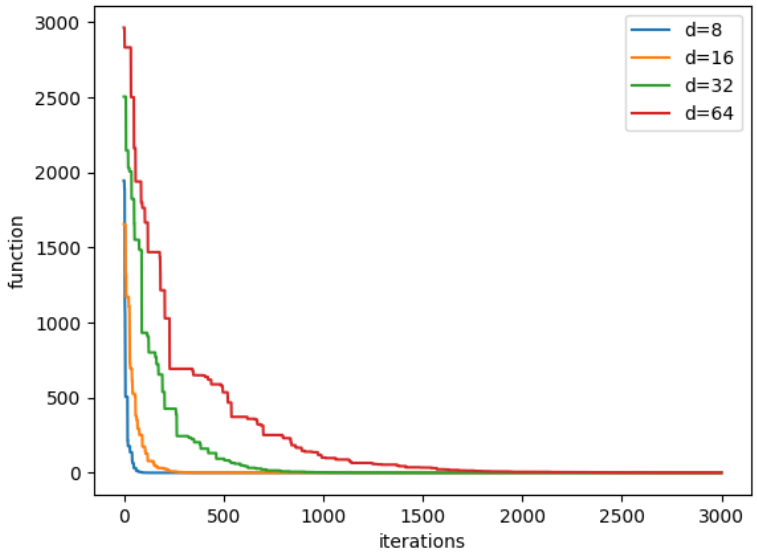


图 6.5: 判别系数随分段数增加的变化

从图中可以看出，随着数据维数不断提高，差分进化算法收敛至全局最优点的迭代数不断增加。正如前文 已知断点数目的多段线性回归 提到的，差分进化算法的收敛速度事实上随着维数的增加而呈指数型增加的趋势。所以，面对维数较大的数据集，比如 Facebook 的经典评论数据集（54 维），该算法很可能开销难以容忍。

6.5 测试 5

对于给定的实例数据集，通过 PLR 类建立模型，考察差分进化算法的收敛速度。这个测试与 测试 4 异曲同工，只是数据集更加具有普适性和一般性。测试的收敛速度如图 6.6 所示。

从图中可以看出，该测试结果与 测试 4 显著不同，对于不同分段数，其最后收敛的值不同，原因是，对于一个真实数据集，不同分段数作用于数据集的效果是不同的。这也为我们找到全局最优分段数提供了思路，我们可以找到具有最小收敛 SS_{res} 值的分段数作为全局最优分段数。但是，这样的想法也是有问题，因为如果 $segment = n$，则 SS_{res} 将等于 0。

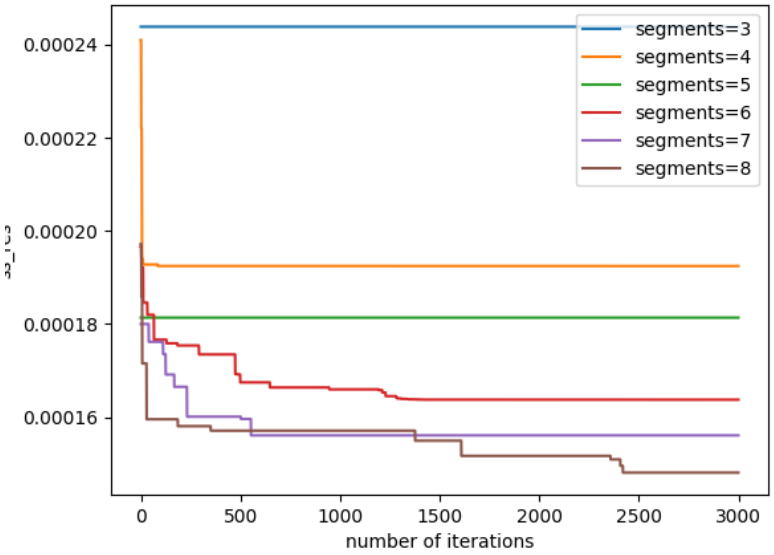


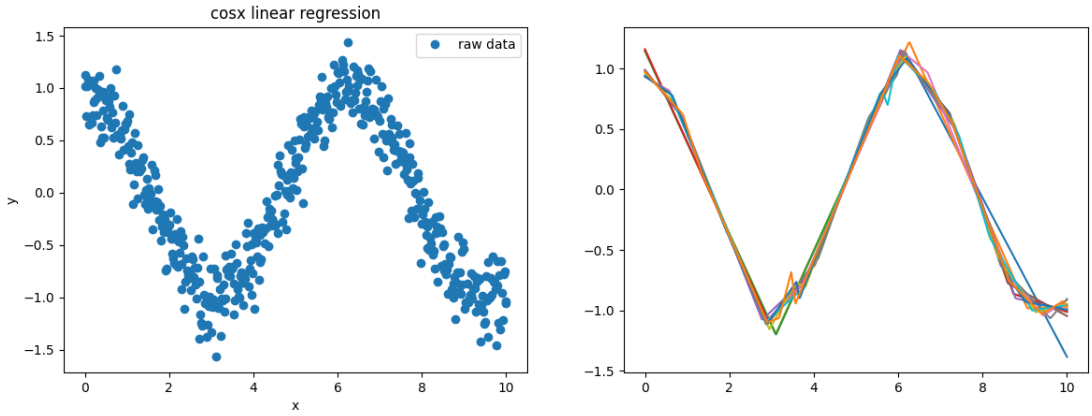
图 6.6: 判别系数随分段数增加的变化

回到图像本身，对于不同的分段数，算法收敛的速度是不一样的，一个显而易见的结论是，最终收敛值越小，所需的迭代次数越多。当然，这其中也有反例，比如 segment7 的收敛速度比 segment6 收敛速度快。因为本实验中采用 $rand/1/bin$ 的策略，所以每次运行的结果也都是不同的。

6.6 测试 6

这个测试中，观察分段线性回归模型对于非多项式函数的拟合情况。采用不同分段数对数据集进行拟合。结果如图 6.7 所示。其中源数据是从函数 $y = cosx$ 进行偏置震荡得来的，如图 7(a) 所示；拟合曲线如图 7(b) 所示。

从图中可以看到，不同分段数对应的拟合曲线差别不大。也就是说，在分段数不大的情况下，拟合的曲线就已经可以反映数据集的大致分布。这给我们的启发是，在实际运用多段线性回归时，不一定需要选择较大的分段数或者全局最优分段数，在较小的分段数下，我们就可以得到曲线的大致分布，并且开销较小（差分进化算法开销随维数增加而指数型增加）。



(a) (b)

图 6.7: 非多项式数据集拟合

6.7 应用结果

分段线性回归的应用场景非常丰富，本次实验选取股票走势拟合进行简单的应用。首先，使用 baostock 第三方库调取的上证指数数据可视化后如图 6.8 所示。观察数据的大致走势，设置好一些列待检的分段数，使用 PLR 类建立模型并进行训练，每个分段数对应的分段函数如 6.9 图所示。

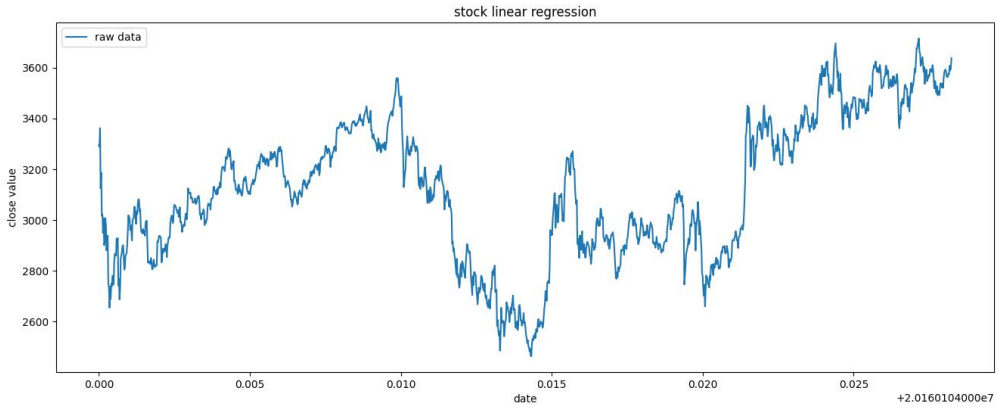


图 6.8: 真实数据：上证指数

从图上可以看出，拟合的大致效果还是不错的，基本上可以反映出上证指数的走势情况。

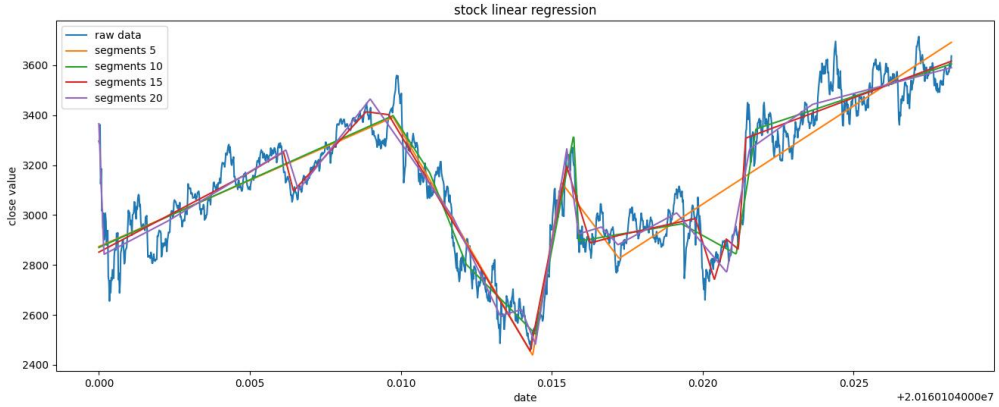


图 6.9: 实际应用：股票拟合

综上所示，本次实验的目的已经达到。面对一个数据集，如果在观察到可以使用多段线性回归来对模型进行拟合的情况下，我们可以选用本次使用中使用的方法。首先是对问题进行建模，通过源数据生成回归矩阵，将多段线性回归问题转换成单一线性回归问题，再使用差分进化算法来对残差平方和进行全局优化得到全局最优的 β 参数，即断点的位置。

7 机器学习课程的学习体会与建议

7.1 项目总结

本次项目的起源是算法课上的一道题目。在课后深入研究过后，我发现这其中好涉及到很多有趣的子问题值得研究。在查阅了相当多的文献之后，我逐渐形成了一个解决问题的思路，通过逐层将问题拆解和规约，我将这个一开始看上去有些棘手的问题转换成了可以逐步求解的问题。

在系统的实现上，我也遵循了我的思维上的分层，将不同问题封装到不同的类中，以此使整个系统更加便于理解，也便于修改和维护。在实现之中，我还增加了数据可视化的模块化功能，这样有助于使用者更进一步研究自己所采用的数据，从数据中得到更多有用的信息，同时也借助数据可视化更进一步理解其中算法的奥秘。

对于差分进化算法，我也是无意中了解到的。一开始我并没有意识到该算法可以用来求解分段线性回归的全局最优解。但是当我了解到分段线性回归可以归约为多维的线性回归，我意识到可以通过差分进化算法优化残差平方和，来使得我最终找到 β 的值，也就是最佳断点位置。

在学习到本项目设计的算法等知识外，我也了解到了非常多的有关 Python 和 Numpy 的奇淫巧计，可以用来快速的解决看似复杂的问题，这在实现层进一步释放了程序员的创造力，让程序员能够有更多精力关注较为上层的东西。我想这样优良的工具应该是计算机科学家或者数据科学家的福音吧。

总而言之，经历了这一个月的思考和摸索以及不断的查阅资料，这个我之前不太明白的问题现在对我来说已经相当简单，我对其中的原理也理解的非常透彻了。

7.2 反思

虽然本次项目的基本目的达到了，但是回首反思这给项目依旧存在诸多可以进一步改进和优化的地方。

首先，最大的一个可以优化的地方就是对于未知断点数目的多段线性回归可以给出一个更加令人满意的解决方案。虽然通过查阅文献，我了解到很多确认断点数目的方法，但是由于平时课业繁重，还有很多算法没有弄明白，因此也没有在系统中实现。

其次，系统中的一些小的细节还有提升的空间，比如数据集的奇异问题，可以通过预处理数据使得数据中不存在两个自变量一样的点来解决，但是由于时间紧急，我也没有来得及修改，只是将奇异异常加入程序，期待使用者能够自行对数据进行处理。

最后，本项目没有考虑自变量多维的情况。即，虽然前期的推导完全适用于多维数据，但是在系统具体实现的过程中，我发现多维数据还需要很多其他的考虑，而因为本项目时间紧急，所以没法把每个细节一一考虑到，因此系统也不支持多维数据的分段线性拟合。

我想，这些还可以优化的细节之处吸引着我进一步在本项目上精进。我想在未来的一段时间里，如果我能抽出时间，我就可以对我上述提的问题进行改进，使得系统整体更加完善。

7.3 学习体会

机器学习这门课是我上大学以来为数不多一些计算机专业的理论实践课。其中涉及到非常多的数学原理和统计学知识，具有理论性，同时机器学习的诸多算法都是非常巧妙高效的，在现实生活中有着非常多应用。

这门课带我领略到潜藏在我们生活中的那些神奇的地方，很多我们习以为常的神奇应用的背后都是有强大的机器学习算法或其它人工智能算法支撑的。也就是说，机器学习这门课如同向我讲述每一个魔法背后的奥秘，带领我掌握其中的奥义，并创造属于自己的神奇应用。

虽然在最后的项目中，我实现的都是一些非常基础的应用，但我想这是我学习机器学习的入门重要一步，在未来我还将继续在这个领域不断钻研学习，争取创造自己神奇应用。

在这门课中我也进一步了解了算法之禅。我一直认为，机器学习和算法是不分家的，机器学习同样包含于算法之中。通过学习机器学习，我了解到算法作为人类智慧结晶的美妙之处，这深深的吸引着我不断向更深处迈进。

7.4 建议

在学完这门机器学习课程之后，我从学生角度对这门课有了一种体验和认知。如果说，这门课还有哪些可以补充的话，我认为，还可以增加最后大项目的选题数。虽然本项目有第三题自选题，但是在缺乏老师的引导下，我自主探索这个课题其实比较艰难，很多时候我还需要找到老师对我加以引导和指正。所以，如果能增加选题数，老师能正面的给予同学们更多的探索方向，我认为是非常好的，这有利于激发同学嗯的探索热情促进同学们更好掌握机器学习模型。

参考文献

[1] Storn, Rainer, and Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces." Journal of global optimization 11.4 (1997): 341-359.

[2] Yang, Lingjian, et al. "Mathematical programming for piecewise linear regression analysis." Expert systems with applications 44 (2016): 156-167.

[3] Malash, Gihan F., and Mohammad I. El-Khaiary. "Piecewise linear regression: A statistical method for the analysis of experimental adsorption data by the intraparticle-diffusion models." Chemical Engineering Journal 163.3 (2010): 256-263.

[4] A. Coppe, R. T. Haftka, and N. H. Kim, "Uncertainty Identification of Damage Growth Parameters Using Nonlinear Regression," AIAA Journal, vol. 49, no. 12, pp. 2818–2821, dec 2011. [Online]. A vailable:

[5] N. Golovchenko, "Least-squares fit of a continuous piecewise linear function," 2004.

[6] 周志华. 机器学习: 第 3 章. 清华大学出版社, 2016.

[7] Yang, Xubing, et al. "Piecewise linear regression based on plane clustering." IEEE Access 7 (2019): 29845-29855.

[8] Muggeo, Vito MR. "Estimating regression models with unknown break-points." Statistics in medicine 22.19 (2003): 3055-3071.