COMP9417 - 机器学习家庭作业1：正则化回归与数值计算优化

引言 在这个作业中，我们将探讨一些基于*梯度*的优化算法。这些算法在过去几十年里对机器学习的发展至关重要。最著名的例子是深度学习中使用的反向传播算法，它实际上只是一种被称为（随机）梯度下降的简单算法的应用。我们将首先在一个确定的问题（没有数据）上从头开始实现梯度下降，然后扩展我们的实现以解决一个现实世界的回归问题。

分数分配 总共有28分。

- 问题1 a):2分
- 问题1 b):1分
- 问题1 c)。4分
- 问题1 d):1分
- 问题1 e):1分
- 问题1 f）。2分
- 问题1 g）。3分
- 问题1 h）。3分
- 问题1 i):1分
- 问题1 j）。4分
- 问题1 k）。5分
- 问题1 l）。1分

提交什么

- **一个单一的PDF**文件，包含每个问题的解决方案。对于每个问题，请以文本和所要求的图表形式提供你的解决方案。对于某些问题，你将被要求提供用于生成答案的代码截图--只有在明确要求的情况下才包括这些截图。
- **包含你在项目中使用的所有代码的.py文件，该文件应在一个单独的.zip文件中提供。**这个代码必须与报告中提供的代码相匹配。

- 如果不遵守这些说明，你可能会被扣分。
- 如果你的作业表述/格式不规范，可能会被扣分。请保持整洁，并使你的解决方案清晰。如有必要，请在新的一页上开始做每个问题。
- 你**不能**提交Jupyter笔记本；这将得到一个零分。但这并不妨碍你在笔记本中开发代码，然后将其复制到.py文件中，或者使用**nbconvert**或类似的工具。
- 我们将建立一个Moodle论坛，用于解答关于这个作业的问题。在发布新问题之前，请阅读现有的问题。在发布问题之前，请在网上做一些基本的研究。请只发布澄清性问题。任何被认为是捕风捉影的问题将被忽略和/或删除。
- 请查看Moodle的公告以了解本规范的更新。您有责任查看有关该规范的公告。
- 请自行完成作业，不要与课程中的其他人讨论你的解决方案。对问题的一般性讨论是可以的，但你必须写出你自己的解决方案，并在你的提交中确认你是否讨论过任何问题（包括他们的名字和zID）。
- 像往常一样，我们监控所有的在线论坛，如Chegg，StackExchange等。在这些网站上发布作业问题等同于抄袭，将导致学术不端行为的发生。
- 你**不能**使用SymPy或任何其他符号编程工具箱来回答推导问题。这将导致相关问题的自动评分为零。你必须手动进行推导。

何时何地提交

- 交稿日期：第四周，2022年**6月20日**星期一**下午5点**前。请注意，论坛在周末将不会被积极监测。
- 逾期提交将招致每天**5%的**处罚，这是**可达到的最高成绩**。例如，如果你的成绩是80/100，但你迟交了3天，那么你的最终成绩将是80−3×5=65。逾期5天以上的提交将得到零分。
- 必须通过Moodle进行提交，没有例外。

问题1.基于梯度的优化

寻找函数f的最小化器的梯度方法的一般框架。RnR的定义如下 →

$$x^{(k+1)} = x^{(k)} - \alpha_k \nabla f(x_k), \qquad k = 0, 1, 2, \ldots,$$

(1) 其中$\alpha_k > 0$被称为步长，或学习率。 考虑下面这个简单的例子

最小化函数$g(x)$=2 ^{x3} + 1。 我们首先注意到，$g\cdot(x) = 3x^2 (x^3 + 1)^{-1/2}$ 。 然后我们需要

选择一个x的起始值，例如$x^{(0)} = 1$。让我们也把步长看作是常数，$\alpha_k = \alpha = 0.1$。然后我们有以下的迭代过程。

$x^{(1)} = x^{(0)} - 0.1 \times 3(x^{(0)})^2 ((x^{(0)})^3 + 1)^{-1/2} =$

0.7878679656440357 $x^{(2)} = x^{(1)} - 0.1 \times 3(x^{(1)})^2 ((x^{(1)})^3 + 1)^{-1/2}$

$= 0.6352617090300827$ $x^{(3)} = 0.5272505146487477$

.

而这一过程一直持续到我们终止算法为止（为了你自己的利益，可以快速练习一下，把它与函数的真正最小值进行比较，即$x_* = 1$）。这个想法适用于具有矢量值输入的函数，这在机器学习中经常发生。例如，当我们最小化一个损失函数时，我们是针对一个权重向量β做的。当我们在每次迭代时，步长是恒定的，这种算法被称为梯度下降。在这个问题的整个过程中，**不要使用任何现有的梯度方法的实现，这样做将导致整个问题的自动得分为零。**

(a) 考虑以下优化问题。

$$\min_{x \in R^n} f(x),$$

其中

$$f(x) = \frac{1}{2} IAx - {}^{\text{bI2}}_{222} + {}^{\gamma}\text{IxI2}。$$

而其中$A \in R^{m\times n}$，$b \in R^m$定义为

$$A = \begin{matrix} & & -1 \\ 1 & 2 & 1 \\ -1 & 10 & 2 \\ 0 & -1 & -21 \end{matrix}, \qquad b = \begin{matrix} 3 \\ 2 \\ -2 \end{matrix},$$

和γ是一个正常数。使用步长α=0.1和γ=0.2，起点$x^{(0)}$ =（1，1，1，1），对f运行梯度下降。当满足以下条件时，你需要终止该算法：$f(x^{(k)})_2 < 0.001$。在你的答案中，清楚地写下这个问题的梯度步骤（1）的版本。同时，打印出x的前5个和后5个值$^{(k)}$，明确指出k的值，形式为。

$$\begin{aligned} k = 0, &\quad x^{(k)} = [1, 1, 1, 1] \\ k = 1, &\quad x^{(k)} = - - - - \\ k = 2, &\quad x^{(k)} = - - - - \end{aligned}$$

:

要提交的内容：一个概述显式梯度更新的方程，打印出你的迭代的前5行（k=5，含）和最后5行。使用四舍五入函数将你的数字四舍五入到小数点后4位。包括本节所用的任何代码的屏幕截图和你在solutions.py中的python代码的副本。

(b) 在上一部分，我们使用了终止条件$f(x^{(k)})_2 < 0.001$。你认为这个条件在算法收敛到f的最小值方面意味着什么？如果把右手边变小（比如说0.0001），会如何改变算法的输出？前面说过。

要提交的内容：一些评论。

(c) 在实验2中，我们介绍了PyTorch并讨论了如何使用它来执行梯度下降。在本题中，你将复制你之前在(a)部分的分析，但用PyTorch代替。和(a)部分一样，清楚地写下这个问题的梯度步骤(1)的版本。同时，打印出x的前5个和后5个值$^{(k)}$，明确指出k的值。如果你觉得有帮助，你可以使用下面的代码作为模板。请注意，你不能在这里对NumPy进行任何调用。

```
进口火炬
import torch.nn as nn
from torch import
optim

A=###
B=###
tol = ###
gamma = 0.2
alpha = 0.1

class
    MyModel(nn.Module):
    def init (self):
        super(). init ()
        self.x = ####

    def forward(self,
        ###): return ###

model = MyModel()
optimizer = ###
terminationCond =
False k = 0
而非终止条款。
    ###计算损失，寻找梯度，更新，检查终止条件等。
```

要提交的内容：打印出你的迭代的前5行（k=5，含）和最后5行。使用round函数将你的数字四舍五入到小数点后4位。包括本节所用的任何代码的屏幕截图和你在solutions.py中的python代码的副本。

在接下来的几个部分，我们将使用上面探讨的梯度方法来解决一个真正的机器学习问题。考虑一下CarSeats.csv中提供的CarSeats数据。 它包含400个观察值，每个观察值都描述了400家商店中的一家出售的儿童汽车座椅。该数据集的特征概述如下。

- 销售额。每个地点的单位销售额（以千计
- CompPrice。竞争者在每个地点收取的价格
- 收入。当地收入水平（以千美元计）
- 广告：广告预算（以千美元计）

- 人口：当地的人口规模（以千计）

- 价格：商店在每个地点收取的价格
- 货架位置：一个分类变量，有坏、好和中等，描述了汽车座椅的货架位置的质量。
- 年龄：当地人口的平均年龄
- 教育。每个地点的教育水平
- 城市 一个分类变量，级别为 "否 "和 "是"，用于描述商店是在城市还是在农村。
- 美国。一个分类变量，级别为No和Yes，用于描述该商店是否在美国。

目标变量是销售额。我们的目标是学习预测作为上述特征子集的函数的销售量。我们将通过运行Ridge回归（Ridge）来实现这一目标，其定义如下

$$\hat{\beta}_{\text{Ridge}} = \arg\min_{\beta} \frac{1}{n} \|y - X\beta\|_2^2 + \varphi\|\beta\|_2^2。$$

其中$\beta \in \mathbb{R}^p$，$X \in \mathbb{R}^{n\times p}$，$y \in \mathbb{R}^n$，$\varphi > 0$。

(d) 我们首先需要对数据进行预处理。删除所有的分类特征。然后使用 `sklearn.preprocessing.StandardScaler`对剩下的特征进行标准化。打印出每个标准化特征的平均值和方差。接下来，将目标变量居中（子 tract its mean）。最后，从所得数据集的前一半创建一个训练集，从剩下的一半创建一个测试集，并称这些对象为X训练、X测试、Y训练和Y测试。打印出每个对象的第一行和最后一行。

提交内容：打印出特征的平均值和方差，打印出4个要求的对象的第一行和最后一行，以及一些评论。包括本节所用的任何代码的屏幕截图和你在solutions.py中的python代码的副本。

(e) 很明显，$\hat{\beta}_{\text{Ridge}}$存在一个封闭式的表达式。 写下这个封闭式表达式，并在φ=0.5的训练数据集上计算出精确的数值。

要提交什么？你的工作，以及基于(X火车，Y火车)的山脊解决方案的数值打印出来。包括本节所用的任何代码的屏幕截图和你在solutions.py中的python代码的副本。

我们现在要解决山脊问题，但要使用数字技术。正如讲座中所指出的，有几个梯度下降的变种，我们将在这里简要地概述一下。回顾一下，在梯度下降法中，我们的更新规则是

$$\beta^{(k+1)} = \beta^{(k)} - \alpha_k \nabla L(\beta^{(k)}), \quad k = 0, 1, 2, \ldots,$$

其中$L(\beta)$是我们试图最小化的损失函数。在机器学习中，通常情况下，损失函数的形式为

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n} L_i(\beta)。$$

即损失是n个函数的平均数，我们将其标记为L_i。因此，梯度也是一个平均数，其形式为

$$\nabla L(\beta) = \frac{1}{n}\sum_{i=1}^{n} \nabla L_i(\beta)。$$

我们现在可以定义一些流行的梯度下降的变体。

(i) 梯度下降（GD）（也被称为批量梯度下降）：在这里，我们使用全梯度，就像我们在所有 n 个项上取平均值一样，所以我们的更新规则是。

$$\beta^{(k+1)} = \beta^{(k)} - \frac{\alpha_k}{n}\sum_{i=1}^{n} \nabla L_i(\beta^{(k)}), \qquad k = 0, 1, 2, \ldots .$$

(ii) 随机梯度下降（SGD）：在第k步，我们从$\{1, \ldots\ldots, n\}$中随机选择一个索引i_k，而不是考虑所有的n个条款。, $n\}$中随机选择一个索引i，并更新

$$\beta^{(k+1)} = \beta^{(k)} - \alpha_k \nabla L_{i_k}(\beta^{(k)}) \qquad ,k = 0, 1, 2, \ldots .$$

这里，我们用$\nabla L_{i_k}(\beta)$来逼近全梯度$\nabla L(\beta)$。

(iii) 小批量梯度下降。GD（使用所有术语）和SGD（使用单一术语）代表了两个可能的极端。在迷你批次GD中，我们在每一步随机选择大小为$1 < B < n$的批次，将其指数称为$\{i_{k_1}, i_{k_2}, \ldots, i_{k_B}\}$，然后我们更新

$$\beta^{(k+1)} = \beta^{(k)} - \frac{\alpha_k}{n}\sum_{j=1}^{B} \nabla L_{k_j}(\beta^{(k)}), \qquad k = 0, 1, 2, \ldots ,$$

因此，我们仍然在逼近全部梯度，但使用的是SGD中的多个单元素。

(f) 岭回归的损失是

$$L(\beta) = \frac{1}{n}\|y - X\beta\|_2^2 + \varphi\|\beta\|_2^2。$$

证明我们可以写出

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n} L_i(\beta)。$$

并确定函数$L_1(\beta), \ldots, L_n(\beta)$。此外，计算梯度$\nabla L_1(\beta), \ldots, \nabla L_n(\beta)$

要提交的内容：你的工作。

(g) 在这个问题中，你将从头开始实现（批处理）GD来解决岭回归问题。使用初始估计值$\beta^{(0)} = 1_p$（1的矢量），和$\varphi = 0.5$，并运行算法1000个epochs（一个epoch是对整个数据的一次传递，所以是一个GD步骤）。对下面的步骤大小重复这一步骤。

$$\alpha \in \{0.000001, 0.000005, 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01\}$$

为了监测算法的性能，我们将绘制数值

$$\Delta^{(k)} = L(\beta^{(k)}) - L(\hat{\beta})。$$

其中$\hat{\beta}$是前面得出的真正的（封闭形式）山脊解。 用3 3网格图展示你的结果，每个子图显示以特定的步长运行GD时$\Delta^{(k)}$的进展情况。说明你认为哪种步长是最好的，并让$\beta^{(K)}$表示在以该步长运

行GD时取得的估计值。报告如下。

(i) 训练的MSE：$\frac{1}{n}\|y_{\text{train}} - X_{\text{train}}\beta^{(K)}\|_2^2$

(ii) 测试MSE：$\frac{1}{n}\|y_{\text{test}} - X_{\text{test}}\beta^{(K)}\|_2^2$

要提交的内容：一个单一的图，要求的训练和测试MSE。包括本节所用的任何代码的屏幕截图和你在solutions.py中的python代码的副本。

(h) 我们现在将从头开始实施SGD，以解决山脊回归问题。使用初始估计值$\beta^{(0)} = 1_p$ （1的矢量）和$\varphi = 0.5$，并运行算法5个epochs（这意味着总共$5n$个β的更新，其中n是训练集的大小）。对下面的步骤大小重复这个过程。

$$\alpha \in \{0.000001, 0.000005, 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.006, 0.02\}$$

像上一个问题一样，提出一个类似的3 3网格图。我们不是在SGD的每一步中随机选择一个指数，而是按照X训练中存储的顺序循环观察，以确保结果的一致性。报告最佳步长的选择以及相应的训练和测试MSE。在某些情况下，你可能会观察到$\Delta^{(k)}$ 的值上下跳动，这不是你使用批处理GD所能看到的。你认为为什么会出现这种情况？

要提交的内容：一个单一的图，要求的训练和测试MSE和一些评论。包括本节所用的任何代码的屏幕截图和你在solutions.py中的python代码的副本。

(i) 根据你的GD和SGD结果，你更喜欢哪种算法？什么时候使用GD比较好？什么时候使用SGD比较好？

(j) 请注意，在GD、SGD和小批量GD中，我们总是在每个迭代中更新整个p维向量β。另一种流行的方法是单独更新每个p参数。为了更清楚地说明这个想法，我们把脊柱损失$L(\beta)$写成$L(\beta_1, \beta_2 \dots, \beta_p)$。我们初始化$\beta^{(0)}$，然后求解$k = 1, 2, 3, \dots,$

$$\beta_1^{(k)} = \arg\min_{\beta_1} L(\beta_1, \beta_2^{(k-1)}, \beta_3^{(k-1)}, \dots, \beta_p^{(k-1)})$$
$$\beta_2^{(k)} = \arg\min_{\beta_2} L(\beta_1^{(k)}, \beta_2, \beta_3^{(k-1)}, \dots, \beta_p^{(k-1)})$$
$$\vdots$$
$$\beta_p^{(k)} = \arg\min_{\beta_p} L(\beta_1^{(k)}, \beta_2^{(k)}, \beta_3^{(k)}, \dots, \beta_p)。$$

请注意，每一个最小化都是在β的一个（一维）坐标上进行的，还有就是只要我们更新$\beta_j^{(k)}$，我们就会在解决$\beta_{j+1}^{(k)}$ 的更新时使用新的值，以此类推。

我们的想法是通过这些坐标级的更新进行循环，直到收敛。在接下来的两部分中，我们将从头开始为里奇回归问题实现这一算法。

$$L(\beta) = \frac{1}{n}\|y - X\beta\|_2^2 + \varphi\|\beta\|_2^2$$

注意，我们可以写出 $n \times p$矩阵$X = [X_1, \dots, X_p]$，其中X_j是X的第j列。寻找优化的解决方案

$$\hat{\beta}_1 = \arg\min L(\beta_1, \beta_2, \dots, \beta_p)。$$

β_1

在此基础上，推导出$\boldsymbol{j} = 2, 3,$ ......的$\hat{\beta}_j$的类似表达。, p.

提示：注意扩展。$X\beta = X_j \beta_j + X_{-j} \beta_{-j}$ ，其中X_{-j} 表示矩阵X，但去掉了第j列，同样β_{-j} 是去掉第j坐标的向量β。*要提交的内容：*你的工作成果。

(k) 在训练数据集上实施前一个问题中概述的算法。在你的实施过程中，一定要按顺序更新β_j ，并使用初始估计值$\beta^{(0)} = 1_p$ （第1个1的矢量），和$\varphi = 0.5$。在10个周期后终止算法（这里的一个周期是p个更新，每个β_j ），所以你将有总共$10p$个更新。报告你得到的模型的训练和测试MSE。在这里，我们想比较三种算法：新的算法与批量GD和SGD的算法，从你以前的答案中选择最佳的步骤大小。像以前一样创建一个k与Δ的关系图$^{(k)}$ ，但这次要绘制三种算法的进展。请确保在你的图中使用与这里相同的颜色，并添加一个图例，清楚地标明每个系列。对于你的批次GD和SGD，在图例中包括步骤大小。你的X轴只需要从$k = 1, . . 10p$.此外，报告你的新算法的训练和测试MSE。*注意：你们中的一些人可能会担心，我们正在比较GD的一个步骤和SGD的一个步骤以及新的算法，我们将暂时忽略这个技术问题。要提交的东西：一个单一的图，要求的训练和测试MSE。*

(l) 在(d)部分，我们将整个数据集标准化，然后分成训练集和测试集。有鉴于此，你认为你在(e)-(k)部分的结果是更可靠、更不可靠，还是不受影响？解释一下。*要提交的内容：你的评论*