COMP9417 - 机器学习

作业2：牛顿法和估计者的平均平方误差

引言 在作业1中，我们考虑了梯度下降法（和坐标下降法）来最小化正则化的损失函数。在这个作业中，我们考虑了另一种被称为牛顿算法的方法。我们将首先在一个简单的玩具问题上运行牛顿算法，然后在一个现实生活中的分类问题上从头开始实现它。我们还将深入研究估计的概念，并比较基于偏差、方差和MSE的估计器。

分数分配 总共有28分。

- 问题1 a):1分
- 问题1 b):1分
- 问题1 c):1分
- 问题2 a):2分
- 问题2 b):2分
- 问题2 c):5分
- 问题2 d):1分
- 问题2 e）。3分
- 问题2 f）。5分
- 问题2 g）。1分
- 问题3 a):3分
- 问题3 b):2分
- 问题3 c):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的屏幕截图--只有在明确要求的情况下才包括这些。
- **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。

- 如果不遵守这些指示，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。你有责任查看关于该规范的公告。
- 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。

何时何地提交

- 交稿日期：第七周，2022年**7月11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.牛顿方法的介绍

注意：在本题中，除非问题中明确要求，否则不要使用所讨论的任何算法的任何现有实现方法。使用现有的实现方式会导致整个问题的成绩为零。在作业1中，我们学习了梯度下降（GD），它通常被称为一阶方法。在这里，我们学习一种被称为牛顿算法的替代算法，它通常被称为二阶方法。粗略地说，二阶方法同时使用了一阶和二阶导数。一般来说，二阶方法要比一阶方法准确得多。给定一个两次可微的函数$g:\mathbb{R}\to\mathbb{R}$，牛顿方法

根据以下更新规则，迭代地生成一个序列$\{x^{(k)}\}$。

$$x^{(k+1)} = x^{(k)} - \frac{糯米(x^{(k)})}{g''(x^{(k)})}, \qquad k = 0, 1, 2, \ldots, \tag{1}$$

例如，考虑函数$g(x) = \frac{1}{2}x^2 - \sin(x)$，初始猜测$x^{(0)} = 0$。

$$g'(x) = x - \cos(x)。\qquad 和 \qquad g''(x) = 1 + \sin(x)。$$

因此，我们有以下的迭代。

$$x^{(1)} = x^{(0)} - \frac{x^{(0)} - \cos(x^{0})}{1 + \sin(x^{(0)})} = 0 - \frac{0 - \cos(0)}{1 + \sin(0)} = 1$$
$$x^{(2)} = x^{(1)} - \frac{x^{(1)} - \cos(x^{1})}{1 + \sin(x^{(1)})} = 1 - \frac{1 - \cos(1)}{1 + \sin(1)} = 0.750363867840244$$
$$x^{(3)} = 0.739112890911362$$

.

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，可以快速编写代码，画出函数和每个迭代的结果）。我们在此注意到，在实践中，我们经常使用一种不同的更新方法，称为*阻尼*牛顿法，其定义为

$$x^{(k+1)} = x^{(k)} - \alpha \frac{g'(x_k)}{g''(x_k)}, \qquad k = o, 1, 2, \ldots. \tag{2}$$

在这里，与GD的情况一样，步长α具有 "抑制 "更新的作用。

(a) 考虑到两次可微的函数$f:\mathbb{R}^n\to\mathbb{R}$。现在，这种情况下的牛顿步骤是。

$$x^{(k+1)} = x^{(k)} - (H(x^{(k)}))^{-1}\,\nabla f\,(x^{(k)})。\qquad k = 0, 1, 2, \ldots, \tag{3}$$

其中$H(x)=\nabla^2 f(x)$是f的Hessian。启发式地解释(用两句话)上述公式是如何将方程(1)推广到具有矢量输入的函数的。 *提交什么？*
一些评论

(b) 考虑函数$f:\mathbb{R}^2\to\mathbb{R}$，定义为

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2。$$

使用`mplot3d`创建该函数的三维图（例子见lab0）。此外，计算f的梯度和Hessian。*要提交的内容。一个单一的图，用于生成该图的代码，计算出的梯度和Hessian，以及所有的工作。在*

solutions.py 中添加一份代码的副本

(c) 只用NumPy，实现（未削弱的）牛顿算法，找到上一部分中函数的最小化，使用初始猜测$x^{(0)}$ $= (-1.2, 1)^T$。终止该算法
当$\|\nabla f(x^{(k)})\|_2 \leq 10^{-6}$。报告$k$=0，1，...的$x$的值$^{(k)}$。，$K$，其中$K$是你的最终
迭代。*要提交的内容：你的迭代，以及你的代码的屏幕截图。在solutions.py中添加一份代码的副本*

问题2.数值化解Logistic回归

注意：在本题中，除非问题中明确要求，否则不要使用所讨论的任何算法的任何现有实现方法。使用现有的实现方式会导致整个问题的成绩为零。在这个问题中，我们将比较梯度下降和牛顿算法来解决逻辑回归问题。回顾一下，在逻辑回归中，我们的目标是最小化对数损失，也被称为交叉熵损失。对于一个截距$\beta_0 \in \mathbb{R}$。
参数向量 $\beta = (\beta_1, ..., \beta_p) \in \mathbb{R}^p$，目标 $y_i \in \{0, 1\}$，和特征向量 $x_i = (x_{i1}, x_{i2}, ..., x_{ip})^T \in \mathbb{R}^p$ 对于$i = 1, ..., n$，我们将使用的（ℓ_2-regularized）对数损失是。

$$L(\beta_0, \beta) = \frac{1}{2}\|\beta\|_2^2 + \frac{\lambda}{n}\sum_{i=1}^{n} y_i \ln\left(\frac{1}{\sigma(\beta_0 + \beta^T x_i)}\right) + (1 - y_i)\ln\left(\frac{1}{1 - \sigma(\beta_0 + \beta^T x_i)}\right),$$

其中$\sigma(z) = (1 + e^{-z})^{-1}$是logistic sigmoid，λ是一个控制正则化程度的超参数。注意，在helper.py中为你提供了这个损失的实现。

(a) 假设你打算用梯度下降法解决这个问题，并选择单独更新每个坐标���为每个分量$\beta_0, \beta_1, ..., \beta$ 的步长α和正则化参数λ推导出梯度下降更新。$, \beta_p$，步长α和正则化参数λ。也就是说，推导出项†的明确表达式。
在以下方面。

$$\beta_0^{(k)} = \beta_0^{(k-1)} - \alpha \times \dagger$$
$$\beta_1^{(k)} = \beta_1^{(k-1)} - \alpha \times \dagger$$
$$\beta_2^{(k)} = \beta_2^{(k-1)} - \alpha \times \dagger$$
$$\vdots$$
$$\beta_p^{(k)} = \beta_p^{(k-1)} - \alpha \times \dagger$$

使你的表达尽可能简单，并确保包括你所有的工作。**提示：如果你还没有这样做，请看一下教程3的第4题。***要提交的内容：你的坐标级GD更新，以及任何工作。*

(b) 对于非截距成分$\beta_1, ..., \beta_p$，以矢量形式重写前一个问题的梯度下降更新，即得出以下项†的明确表达式。

$$\beta^{(k)} = \beta^{(k-1)} - \alpha \times \dagger$$

你的表达式应该只用β_0, β, x_i 和 y_i。接下来，让$\gamma = [\beta_0, \beta^T]^T$ 是结合截距和系数向量β的（p + 1）维向量，写下更新

$$\gamma^{(k)} = \gamma^{(k-1)} - \alpha \times \dagger。$$

注意：这个最终表达式将是我们对梯度下降的矢量化实现。上述练习的重点只是要注意截距和非截距参数之间的区别。在坐标上做GD在实践中是非常低效的。*要提交的东西：你的矢量化GD更新以及任何工作。*

(c) 推导出步长为α的逻辑回归问题的（减弱的）牛顿更新。请确保包括你所有的工作。为了便于记述，你可能会发现写成$z^{(k)} = \beta_0^{(k)} + x_i\beta^{T(k)}$是有帮助的。**提示：当做对于这个问题，首先要解决的是没有正则化项的问题（即忽略山脊惩罚项，取λ=1）。一旦你有了这种情况下的Hessian的形式，将它扩展到这里所需要的更普遍的设置应该是更直接的。** *要提交的东西：你的矢量阻尼牛顿更新以及任何工作。*

(d) 现在，我们将使用前几部分得出的更新数据，在一个真实的数据集上比较GD和牛顿算法的性能。要做到这一点，我们将使用 `songs.csv` 数据集。该数据包含各种歌曲的信息，还包含一个概述歌曲流派的类变量。如果你有兴趣，你可以在这里阅读更多关于该数据的信息，尽管对每一个特征的深入理解对于本评估的目的来说并不重要。装入数据并进行以下预处理。

(I) 删除以下功能。"艺术家姓名"、"曲目名称"、"调性"、"模式"、"时间特征"、"器乐性"

(II) 目前的数据集有10个类，但我们在这里描述的逻辑回归的形式只适用于二元分类。我们将把数据限制在5类（hiphop）和9类（pop）。在删除其他类后，重新对变量进行编码，使目标变量对hiphop来说是y=1，对pop来说是y=0。

(III) 删除任何特征值缺失的剩余行。你剩下的数据集应该总共有3886行。

(IV) 使用`sklearn.model selection.train test split`函数，将你的数据分成X训练、X测试、Y训练和Y测试。使用测试大小为0.3，随机状态为23，以保证可重复性。

(V) 将`sklearn.preprocessing.StandardScaler`适合于所得到的训练数据，然后使用这个对象来扩展你的训练和测试数据集。

(VI) 打印出X训练、X测试、Y训练、Y测试的第一行和最后一行（但只有X训练、X测试的前三列）。

要提交的东西：（VI）中要求的行的打印结果。你在solutions.py中的代码副本

一个简单的旁证。回溯式直线搜索。在作业1中，我们在实现中天真地选择了步长参数，通过观察步长网格并选择最佳步长。在实践中，这显然是不可行的（为大量的候选步长多次训练模型的计算成本）。一种非常流行的、经验上成功的方法被称为回溯线搜索（BLS）。在BLS中，步长是在模型的每个迭代中自适应选择的。

通过检查一个给定的标准来确定算法。特别是，选择参数$0 < a \leq 0.5$和$0 < b < 1$。在算法的迭代k（我们正在最小化函数f（x）），当前迭代为$x^{(k)}$，设置步长为$\alpha = 1$。然后，虽然

$$f(x^{(k)} - \alpha \nabla f(x^{(k)})) > f(x) - a\alpha \| \nabla f(x^{(k)}) \|_2^2$$

根据$\alpha = b\alpha$缩小步长，否则在你的更新中使用当前步长α。我们不会在这里解释为什么这样做是明智的，但如果你有兴趣，你可以在网上找到相关信息。(当然，我们也可以在咨询会议中讨论这个问题）。然而，重要的是，BLS的思想可以应用于梯度下降和阻尼牛顿算法，在下面的问题中，你将被要求在这两种算法的实现中使用BLS的方法。

(e) 在训练数据集上运行梯度下降与回溯线搜索，时间为60个历时，λ=0.5，并将$\beta^{(0)}=0$，$\beta^{(0)}=0_p$，其中0_p是p维的零点矢量。对于你的BLS

实施时取a=0.5，b=0.8。报告你的训练和测试损失，以及每次迭代的步长和每次迭代的训练损失图。**提示：如果你需要一个理智的检查，在这里。**

最好的办法是用sklearn来拟合逻辑回归模型。这应该让你知道你的实现应该达到什么样的损失（如果你的实现做得一样好或更好，那么你就在正确的轨道上）。*要提交什么：两张图，一张是步长，另一张是训练损失。报告你的火车和测试损失，以及本节中使用的任何代码的屏幕截图，还有你在solutions.py中的代码副本。*

(f) 在训练数据集上运行带有回溯线搜索的阻尼牛顿算法60个历时，λ=0.5。对你的BLS算法使用与前一个问题相同的参数。使用与前一个问题相同的初始化β_0，β 报告你的训练和测试损失，以及你在每个迭代中的步长的图。此外，提供一个单一的图，显示GD和牛顿算法的训练损失（使用标签/图例使你的图易于阅读）。 *提交内容：两张图，一张是步长，另一张是训练损失（GD和牛顿）。报告你的训练和测试损失，以及本节中使用的任何代码的屏幕截图，还有你在solutions.py中的代码副本。*

(g) 一般来说，事实证明牛顿的方法要比GD好得多，事实上牛顿算法的收敛性是二次的，而GD的收敛性是线性的（比二次慢得多）。鉴于此，你认为为什么梯度下降及其变种（如SGD）在解决机器学习问题方面要受欢迎得多？*提交什么：一些评论*

问题3.关于MLE的更多信息

让$X_1, \ldots, X_n \sim N(\mu, \sigma^2)$.回顾一下，在教程2中，我们表明$\mu$、$\sigma$的MLE估计值^2。$\hat{\mu}_{\text{MLE}}$和$\hat{\sigma}^2_{\text{MLE}}$其中

$$\hat{\mu}_{\text{MLE}} = \overline{X}。 \qquad 和 \qquad \hat{\sigma}^2_{\text{MLE}} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2。$$

在这个问题上，我们将更深入地探讨这些估计器。

(a) 求$\hat{\mu}_{MLE}$ 和$\hat{\sigma}^2_{MLE}$ 的偏差和方差。**提示：**你可以不加证明地使用以下事实

$$\text{变化}\left(\frac{1}{\sigma^2}\sum_{i=1}^{n}(X_i - \overline{X})^2\right) = 2(n-1)$$

要提交的内容：估计器的偏差和方差，以及你的工作。

(b) 你的朋友告诉你，他们有一个更好的σ估计器^2，即。

$$\tilde{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2 \text{。}$$

讨论一下这个估计器是比MLE估计器好还是坏。

$$\hat{\sigma}^2_{\mathrm{MLE}} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2 \text{。}$$

一定要包括对两个估计者的偏差和方差的详细分析，并描述随着样本量n*的*增加，这些量（对每个估计者）会发生什么变化（使用图表）。对于你的图表，你可以假设σ=1。

要提交的内容：新估计器的偏差和方差。比较两个估计器的偏差与样本量n的关系的图，比较两个估计器的方差与样本量n的关系的图，在你的图中使用标签/图例。在solutions.py中使用的代码的副本

(c) 计算并绘制上一部分所考虑的两个估计者的MSE。对于你的图表，你可以假设σ=1。*提供*一些讨论，说明哪个估计器更好（根据它们的MSE），以及当样本量n变大时会发生什么。*比较估计器的MSE与样本量n的关系的图，以及一些评论。在你的图中使用标签/图例。在这里使用的solutions.py的代码的副本*