基于统计方法的泛化学习初探

薛翔元

目录

摘要

人工智能是当今时代的关键科学领域，其研究的热门在于机器学习、深度学习等算法。在机器学习中，迁移学习是一类特殊的任务，旨在充分利用已经训练完成的模型，通过迁移算法使其仍然适用于其他数据集。模型泛化是迁移学习的进阶任务，要求模型从不满足独立同分布假设的数据集上挖掘数据共有的本质特征，达到在任意数据分布下均为最优的效果。

本文避开机器学习中视觉或自然语言的具体任务，将问题简化到线性形式，以探究泛化学习算法的本质。本文将线性回归问题作为基本研究目标，在经典的线性回归上做出修改，在数据集上额外构造一维数据作为虚假特征。同时，我们将数据拆分为多个组别，使数据在组内独立同分布，在组间存在分布差异。我们试图设计一种泛化学习算法，使其能够避免虚假特征的干扰，得到正确的线性回归结果。

首先，我们推导了最大似然估计和最小二乘线性回归的数学公式，并依据此结果设计了传统的梯度下降求解算法。接着，我们尝试利用 Python 实现传统算法，在数据集上进行线性回归。然而，虚假特征的存在严重干扰了传统线性回归算法的正常求解，使算法产生了错误的回归结果。此外，我们通过实验发现，模型在数据集上更趋向于习得虚假的线性关系。我们分别尝试了按真实特征、虚假特征和混合特征进行回归，得到的结果对后续的实验具有参考意义。然后，我们根据论文利用概率统计方法，从理论上推导了经验风险最小化的公式，并设计了线性回归任务中的损失函数。我们利用 PyTorch 实现了泛化学习算法，并在数据集上验证了线性回归效果。实验表明泛化算法可以排除虚假特征的干扰，得到正确的回归结果。最后，我们在更大规模的数据集上测试了泛化算法的回归效果，并利用统计方法进行了分析。

此外，我们对本文中运用的概率统计相关研究方法进行了总结，并指出了本文研究的价值。概率论与数理统计的方法将在泛化学习的研究中发挥更多重要的作用。

关键字：　概率统计　机器学习　泛化学习

1 问题阐述

1.1 问题背景

1.1.1 独立同分布假设

独立同分布（Independent and Identically Distributed）假设是机器学习的重要理论基础。[3] 该假设认为，训练集和测试集中所有的数据和标签具有独立且相同的分布。也即对于数据集

$$\{(\boldsymbol{x_i}, y_i)|i = 1, 2, \ldots, n\} \tag{1}$$

总有

$$P\left[(\boldsymbol{x_i}, y_i)|(\boldsymbol{x_j}, y_j)\right] = P\left[(\boldsymbol{x_i}, y_i)\right], 1 \leq i, j \leq n \tag{2}$$

以及

$$F\left[(\boldsymbol{x_i}, y_i)\right] = F\left[(\boldsymbol{x_j}, y_j)\right], 1 \leq i, j \leq n \tag{3}$$

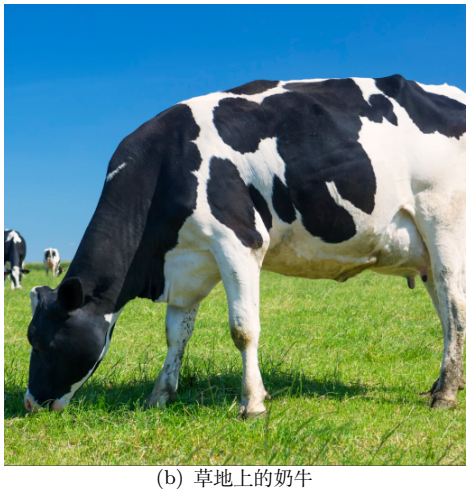
机器学习将模型视为参数系统 $\boldsymbol{\theta}$，将预测视为该系统对数据 $\boldsymbol{x}$ 的观测结果 $\hat{y}$，得到后验概率 $P(\hat{y}|\boldsymbol{\theta}, \boldsymbol{x})$，依据贝叶斯公式重新估计模型参数 $\boldsymbol{\theta}'$，反复迭代以训练模型 $\boldsymbol{\theta}^*$。因此，独立分布保证了参数估计的正确性，相同分布使每个数据都具有代表性，因此模型可以基于当前数据进行训练，用于未来数据的预测。因此，独立同分布假设被广泛应用于几乎所有机器学习模型中。[3]

1.1.2 模型泛化能力

一般来说，简单随机抽样可以直接保证数据集的独立同分布。然而，数据集都是自然界采样的结果，在内容的分布上难免出现差异和依赖关系。此时，如果我们仍然采用独立同分布假设训练模型，将会出现灾难性的结果。

考虑一个图像分类模型。由于自然环境的原因，所有的骆驼都生活在沙漠中，所有的奶牛都生活在草地上。因此，模型训练集的分布存在显著差异，并不满足独立同分布假设。

(a) 沙漠中的骆驼



(b) 草地上的奶牛

图 1: 图像分类任务案例

我们对测试集进行一些构造，使所有的骆驼出现在草地上，所有的奶牛出现在沙漠中，此时模型对两者的分类错误率几乎达到 100%，这是因为独立同分布假设误导了参数估计，使背景成为了动物 “核心特征” 的一部分。[6, 7]

由此可见，模型对特征的提取是狭隘的，缺乏不同分布下的普适性。我们把模型免受分布差异干扰，提取关键特征的能力称为“泛化能力（Generalization）”。模型泛化能力的提升，是当今机器学习领域的研究前沿之一，解决问题的关键在于数理统计和优化控制方法。[6]

1.2 研究目标

1.2.1 泛化学习方法

多层神经网络的复杂模型参数量大且高度非线性，因此我们避免对于这类复杂模型的讨论，而着眼于线性情形。我们试图去寻找某种方法，使模型的参数估计能够免于虚假因果关系的干扰，在独立同分布条件缺失的情况下仍能有较好的预测表现。

为实现这种泛化学习方法，我们依据如下步骤进行探究：1. 传统算法的复现与效果评估；2. 理论探讨与模型改进；3. 改进效果验证与总结。其中，传统算法源于数理统计中的最大似然估计和回归分析，模型改进也基于类似思想，对于模型效果的分析则采用课内常见的统计方法。

1.2.2 实验案例

为保证问题在课程研究的范围内，我们尽可能地简化问题。将所有的数据和标签限定在一维情形，有 $(x_i, y_i) \in \mathbb{R}^2$。依据自然规律，假定随机变量 X 服从正态分布

$$X \sim N(0, \sigma^2) \tag{4}$$

构造随机变量 Y 在相同正态分布误差内与 X 存在线性关系

$$Y = aX + b + \varepsilon, \varepsilon \sim N(0, \sigma^2) \tag{5}$$

再依赖随机变量 Y 构造虚假因果

$$X' = Y + \varepsilon', \varepsilon' \sim N(0, 1) \tag{6}$$

因此，训练集共有三个维度

$$S = \{(x_i, x_i', y_i) | i = 1, 2, \ldots, n\} \tag{7}$$

在本文的研究中，我们取不同的 σ^2 以构造分布不同的样本数据，形成不满足独立同分布假设的训练数据集。泛化能力强的模型可以发现不同的数据分布，排除分布层面的特征差异，提取训练集共有的普遍特征。[6]

我们可以观察评估模型对数据的回归效果，以评价模型的泛化能力。

2 传统算法

2.1 基础理论

2.1.1 最大似然估计

在机器学习中，最大似然估计是参数估计的重要方法，这一理论在教材中有详细介绍。[1, 2] 假设某连续型随机变量 X 的概率密度为

$$f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{\theta} \in \mathbb{R}^k \tag{8}$$

对于样本的一组观测值 $(x_1, x_2, \ldots, x_n)$，似然函数

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} f(\boldsymbol{x}_i; \boldsymbol{\theta}) \tag{9}$$

求解似然函数 $L(\boldsymbol{\theta})$ 的极大值点 $\hat{\boldsymbol{\theta}}$，使

$$L(\hat{\boldsymbol{\theta}}) = \max_{\boldsymbol{\theta} \in \mathbb{R}^k} L(\boldsymbol{\theta}) \tag{10}$$

两侧取自然对数

$$\ln L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ln f(\boldsymbol{x}_i; \boldsymbol{\theta}) \tag{11}$$

求解方程组

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} = 0, i = 1, 2, \ldots, k \tag{12}$$

即得参数 $\boldsymbol{\theta}$ 的最大似然估计 $\hat{\boldsymbol{\theta}}$。

2.1.2 最小二乘线性回归

假设 y 与 $\boldsymbol{x} \in \mathbb{R}^n$ 具有线性关系

$$y = \boldsymbol{\theta}^T \boldsymbol{x} + \varepsilon \tag{13}$$

其中 ε 为误差项，服从正态分布

$$\varepsilon \sim N(0, \sigma^2) \tag{14}$$

也即

$$P(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\varepsilon^2}{2\sigma^2}\right) \tag{15}$$

因此

$$P(y|\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{\left(y - \boldsymbol{\theta}^T \boldsymbol{x}\right)^2}{2\sigma^2}\right] \tag{16}$$

似然函数

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{\left(y_i - \boldsymbol{\theta}^T \boldsymbol{x}_i\right)^2}{2\sigma^2}\right] \tag{17}$$

两侧取自然对数

$$\ln L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ln \left\{ \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{\left(y_i - \boldsymbol{\theta}^T \boldsymbol{x}_i\right)^2}{2\sigma^2}\right] \right\} \tag{18}$$

$$= n \ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y_i - \boldsymbol{\theta}^T \boldsymbol{x}_i\right)^2 \tag{19}$$

最大化对数似然函数，只要最小化平方和函数

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \boldsymbol{\theta}^T \boldsymbol{x}_i\right)^2 \tag{20}$$

上述推导从概率统计角度解释了最小二乘的合理性。[2, 3]

多元线性回归参数 $\boldsymbol{\theta}$ 可以通过正规方程组求解，该方法在课本中有详细介绍。在机器学习中，也可以直接将残差平方和 $J(\boldsymbol{\theta})$ 作为损失函数，通过优化算法求解近似参数。在本文的研究方法中，我们主要采用机器学习中的梯度下降算法进行多元线性回归方程的求解。

2.1.3 损失函数与梯度下降

我们将残差平方和函数作为损失函数

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \boldsymbol{\theta}^T\boldsymbol{x}_i\right)^2 \tag{21}$$

其中

$$\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)^T \tag{22}$$

为该损失函数的决策变量。[3, 4]

注意到每个平方项 $\left(y_i - \boldsymbol{\theta}^T\boldsymbol{x}_i\right)^2$ 均为 $\boldsymbol{\theta}$ 的凸函数，且 $J(\boldsymbol{\theta})$ 为凸函数的线性组合，因此 $J(\boldsymbol{\theta})$ 也为凸函数，具有良好的全局优化性质，使多元线性回归问题转化为凸优化问题

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^k} J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \boldsymbol{\theta}^T\boldsymbol{x}_i\right)^2 \tag{23}$$

利用梯度下降算法可以有效求解该问题。首先，我们任取 $\boldsymbol{\theta}_0 \in \mathbb{R}^k$ 作为 $\boldsymbol{\theta}$ 的初始值；接着，设定步长初始值 t_0，利用回溯线搜索以比例系数 β 缩小步长，得到最优步长 t；然后，将 $\boldsymbol{\theta}$ 沿着梯度方向更新为 $\boldsymbol{\theta} - t\nabla J(\boldsymbol{\theta})$，迭代上述过程直至 $||\nabla J(\boldsymbol{\theta})||$ 充分小。可以证明，梯度下降算法求解的多元线性回归参数可以达到任意精度。[4]

2.2 实验验证

2.2.1 算法实现

考虑本文的具体实验案例，由于样本数据中存在 X, X' 两个特征，我们建立二元线性回归模型

$$y = \alpha x + \beta x' + \gamma \tag{24}$$

根据 x_i, x_i' 的不同权重 α, β，存在三种回归模式

- $\alpha > 0, \beta = 0$：完全根据原始关系回归
- $\alpha = 0, \beta > 0$：完全根据虚假因果回归
- $\alpha > 0, \beta > 0$：均衡权重回归

从直觉上对回归结果进行预测：当我们根据原始关系回归时，显然可以得到正确的结果；如果按照虚假因果回归，则所得参数被完全误导，产生错误的回归结果；当均衡权重回归时，参数估计仍然受到虚假因果的干扰，产生较大的偏差。[6]

根据该回归模型，对于每个数据 $\boldsymbol{x}_i$ 应有 $\boldsymbol{x}_i \in \mathbb{R}^2$，两个分量分别代表真实特征和虚假特征。考虑偏差项 γ，为使数据集以整体的形式参与矩阵运算，我们对 $\boldsymbol{x}_i$ 进行增广，定义

$$\widetilde{\boldsymbol{x}}_i = \begin{pmatrix} \boldsymbol{x}_i \\ 1 \end{pmatrix} \tag{25}$$

此时 $\widetilde{\boldsymbol{x}}_i \in \mathbb{R}^3$，数据集矩阵 $\boldsymbol{X}$ 可以与参数向量 $\boldsymbol{\theta} = (\alpha, \beta, \gamma)^T$ 相乘，回归结果即为

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\theta} \tag{26}$$

这一处理不仅简化了计算，也揭示了偏差项 γ 的本质即为一个新的特征维度。对于任意的 $\lambda \neq 0$，都可以通过数据的升维完成等价的线性回归。因此不失一般性，我们令偏差项 $\lambda = 0$，求解参数 $\boldsymbol{\theta} = (\alpha, \beta)^T \in \mathbb{R}^2$，数据 $\boldsymbol{X} \in \mathbb{R}^{n\times 2}$ 的线性回归模型。[5]

下面，我们利用 Python 编程对实验案例进行线性回归分析，以验证猜测。根据前文的讨论，我们采用梯度下降算法求解线性回归参数，其中，损失函数选用残差平方和函数 $J(\boldsymbol{\theta})$，学习率恒定为 10^{-3}，不考虑权重衰减。依据如下分布规则生成数据集

$$\begin{aligned} X &\sim N(0,\sigma^2) \\ Y &= 2X+\varepsilon, \varepsilon \sim N(0,\sigma^2) \\ X' &= Y+\varepsilon', \varepsilon' \sim N(0,1) \end{aligned} \tag{27}$$

两组数据各包含 10000 个样本，其中第一组数据 $\sigma_1^2=1$，第二组数据 $\sigma_2^2=0.01$，算法描述如下

算法 1 传统最小二乘线性回归

输入： $\boldsymbol{X}\in\mathbb{R}^{n\times 2}$, $\boldsymbol{y}\in\mathbb{R}^n$

输出： $\boldsymbol{\theta}\in\mathbb{R}^2$

$\boldsymbol{\theta}\leftarrow(1,1)^T$

for $k\leftarrow 1$ **to** m **do**

  SHUFFLE($\boldsymbol{X}$, $\boldsymbol{y}$)

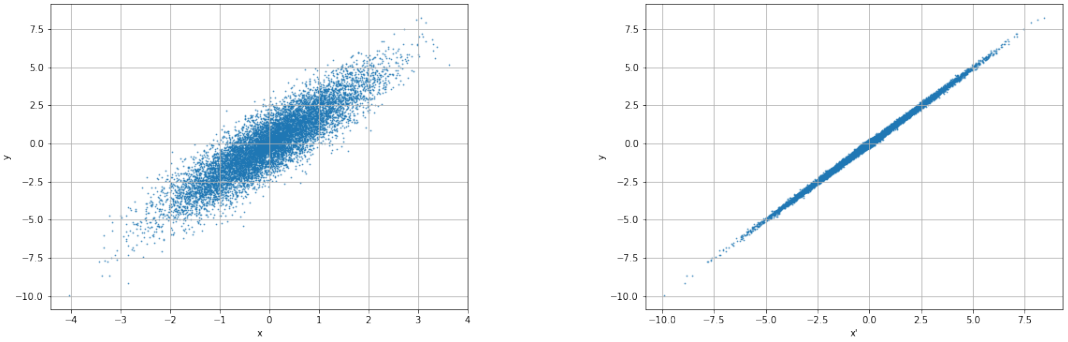
  $J(\boldsymbol{\theta})\leftarrow\frac{1}{n}||\boldsymbol{X\theta}-\boldsymbol{y}||^2$

  $\boldsymbol{\theta}\leftarrow\boldsymbol{\theta}-t\cdot\frac{\partial J(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}$

return $\boldsymbol{\theta}$

2.2.2 实验结果

绘制生成数据的散点图，Y 关于 X 和 X' 的分布情况如图所示



(a) Y 关于 X 的分布　　(b) Y 关于 X' 的分布

图 2: 生成数据分布情况

可以看到，Y 关于 X 和 X' 具有明显的线性关系，其中 $Y\approx 2X\approx X'$，我们希望算法求解出 $y=2x$ 的真实关系回归结果，而非 $y=x$ 的虚假关系回归结果。

通过限定 $\alpha>0,\beta=0$，我们人为消除了虚假特征 X' 的影响，使模型只按真实特征 X 进行回归，得到如下线性回归方程

$$y=2.0097110271453857x\approx 2x \tag{28}$$

与真实线性关系符合良好。

将散点和直线共同绘制，线性回归结果如图所示

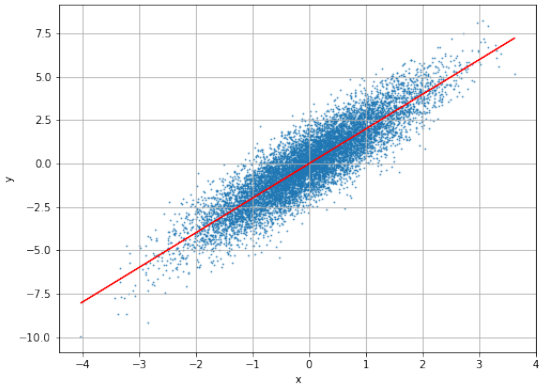


图 3: $\alpha > 0, \beta = 0$ 线性回归结果

该结果完全正确，可以作为其他情形的标准正参考。

通过限定 $\alpha = 0, \beta > 0$，我们强制模型忽视真实特征 X，只按虚假特征 X' 进行回归，得到如下线性回归方程

$$y = 0.995421826839447x \approx x \tag{29}$$

与虚假线性关系完全吻合，从而偏离真实线性关系。

将散点和直线共同绘制，线性回归结果如图所示

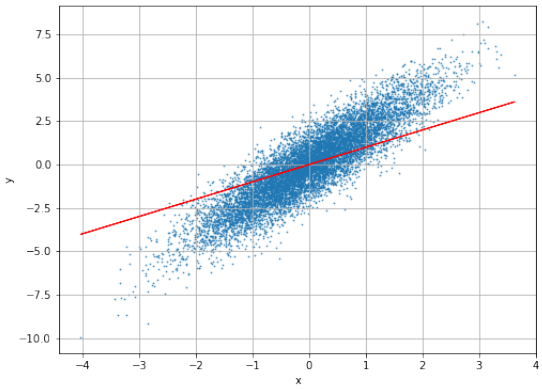


图 4: $\alpha = 0, \beta > 0$ 线性回归结果

该结果可以作为其他情形的标准负参考。

现在，我们令 $\alpha > 0, \beta > 0$，不做任何限制，模型可以自由地从数据集 $\boldsymbol{X}$ 中提取任何特征，真实特征 X 与虚假特征 X' 具有相同的权重，在限制迭代次数的条件下，得到如下线性回归方程

$$y = 0.03915265575051308x + 0.979917585849762x' \tag{30}$$

可以预见，当迭代次数继续增大，线性回归方程收敛于

$$y = x \tag{31}$$

我们尝试解释这一现象：由于 X' 分布的方差较小，完全按照 X' 进行回归可以使损失函数 $J(\boldsymbol{\theta})$ 最小，因此模型最终收敛于虚假的线性关系。[3]

为体现线性回归结果与标准正参考的偏离程度，我们将数据散点和降维直线 $y = \alpha x$ 共同绘制，而忽略 $\beta x'$ 在另一个维度的影响，结果如图所示

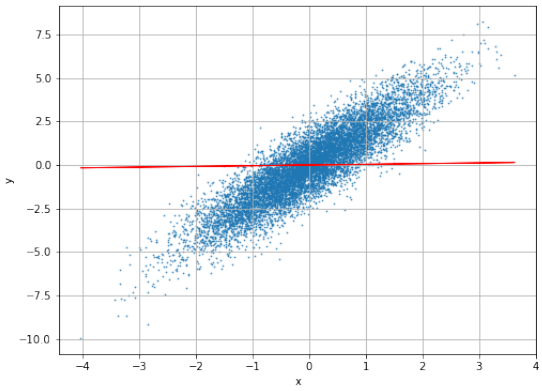


图 5: $\alpha > 0, \beta > 0$ 线性回归结果

可以看到，真实特征 X 的权重几乎为 0，模型对真实的线性关系几乎没有任何回归效果，完全被虚假的线性关系所欺骗。

通过上述实验可知，传统算法对常规的线性回归具有良好的求解效果，但当数据不满足独立同分布假设，存在虚假因果关系时，传统模型缺乏足够的泛化能力，从而难以得到正确的结果。这一问题要求我们设计泛化能力更强的算法模型，从不同的数据分布中挖掘真实的线性关系。

3 泛化算法

3.1 理论推导

3.1.1 优化模型搭建

相比传统的经验风险最小化（Empirical Risk Minimization），Martin Arjovsky 等人在 2019 年提出不变性风险最小化（Invariant Risk Minimization），用以解决非独立同分布条件下模型泛化能力不足的问题。该方法基于数学推导，给出了一种新的泛化学习方式。我们将在本文中运用概率统计的方法重新推导并复现这一模型，在简化的线性情形中验证模型的泛化能力。[6, 7]

为了增强模型的泛化能力，我们需要充分利用不同分布数据之间的差异，以排除虚假的因果关系。因此，我们对数据按照分布划分为若干子集

$$\boldsymbol{X} = \bigcup_{k=1}^{s} \boldsymbol{X}_k \tag{32}$$

使得

$$\boldsymbol{X}_i \cap \boldsymbol{X}_j = \varnothing, \quad \forall 1 \leq i < j \leq s \tag{33}$$

此时，每个子集 $\boldsymbol{X}_i$ 内的数据独立同分布，子集间的数据分布各不相同。定义风险函数

$$R(\boldsymbol{\theta}) = \max_{\boldsymbol{X}_i \subset \boldsymbol{X}} J(\boldsymbol{\theta}) \tag{34}$$

子集 $\boldsymbol{X}_i$ 中 $J(\boldsymbol{\theta})$ 定义为模型参数 $\boldsymbol{\theta}$ 预测均方误差的期望

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\frac{1}{n_k}\sum_{i=1}^{n_k}\left(y_i - \boldsymbol{\theta}^T \boldsymbol{x}_i\right)^2\right] \tag{35}$$

我们希望模型具有泛化能力，也就需要找到一个压缩矩阵 $\boldsymbol{\Phi}:\boldsymbol{X}\to\boldsymbol{\Lambda}$ 和一个回归矩阵 $\boldsymbol{\omega}:\boldsymbol{\Lambda}\to\boldsymbol{y}$，解决如下优化问题

$$\begin{aligned}&\min_{\substack{\boldsymbol{\Phi}:\boldsymbol{X}\to\boldsymbol{\Lambda}\\ \boldsymbol{\omega}:\boldsymbol{\Lambda}\to\boldsymbol{y}}}\sum_{\boldsymbol{X}_i\subset\boldsymbol{X}}J(\boldsymbol{\omega}\circ\boldsymbol{\Phi})\\ &\text{s.t. }\ \boldsymbol{\omega}\in\arg\min_{\boldsymbol{\omega}^*:\boldsymbol{\Lambda}\to\boldsymbol{y}}J(\boldsymbol{\omega}^*\circ\boldsymbol{\Phi}),\forall\boldsymbol{X}_i\subset\boldsymbol{X}\end{aligned}\tag{36}$$

该优化问题存在两个决策变量 $\boldsymbol{\Phi}$ 和 $\boldsymbol{\omega}$，这使求解变得极为困难，因此我们试图将约束条件转化为惩罚项，定义新的风险函数

$$R(\boldsymbol{\Phi},\boldsymbol{\omega})=\sum_{\boldsymbol{X}_i\subset\boldsymbol{X}}J(\boldsymbol{\omega}\circ\boldsymbol{\Phi})+\lambda\cdot\mathbb{D}(\boldsymbol{\Phi},\boldsymbol{\omega})\tag{37}$$

其中，λ 作为系数平衡误差项和惩罚项，$\mathbb{D}$ 为待定函数，用于评估回归矩阵 $\boldsymbol{\omega}$ 的最优性。我们将在下一节详细讨论评估函数 $\mathbb{D}$ 的选取和损失函数的总体设计。[6, 7]

3.1.2 损失函数设计

此前，我们已经把问题简化到线性情形，使 $\boldsymbol{\Phi}$ 和 $\boldsymbol{\omega}$ 可以用矩阵表示，因此我们可以将模型的预测结果 $\hat{\boldsymbol{y}}$ 写成矩阵相乘的形式

$$\hat{\boldsymbol{y}}=\boldsymbol{\omega}\circ\boldsymbol{\Phi}(\boldsymbol{X})=X\boldsymbol{\Phi}\boldsymbol{\omega}\tag{38}$$

对于确定的压缩矩阵 $\boldsymbol{\Phi}$，最优的 $\boldsymbol{\omega}_{\boldsymbol{\Phi}}$ 使方程的残差平方和最小，也即

$$\boldsymbol{\omega}_{\boldsymbol{\Phi}}=\arg\min_{\boldsymbol{\omega}}||\hat{\boldsymbol{y}}-\boldsymbol{y}||\tag{39}$$

因此，该 $\boldsymbol{\omega}_{\boldsymbol{\Phi}}$ 即为方程的最小二乘解

$$\boldsymbol{\omega}_{\boldsymbol{\Phi}}=\mathbb{E}\left[\boldsymbol{\Phi}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\Phi}\right]^{-1}\mathbb{E}\left[\boldsymbol{\Phi}^T\boldsymbol{X}^T\boldsymbol{y}\right]\tag{40}$$

整理得到

$$\mathbb{E}\left[\boldsymbol{\Phi}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\Phi}\right]\boldsymbol{\omega}_{\boldsymbol{\Phi}}-\mathbb{E}\left[\boldsymbol{\Phi}^T\boldsymbol{X}^T\boldsymbol{y}\right]=\boldsymbol{0}\tag{41}$$

因此我们可以定义如下评估函数

$$\mathbb{D}(\boldsymbol{\Phi},\boldsymbol{\omega})=\left|\left|\mathbb{E}\left[\boldsymbol{\Phi}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\Phi}\right]\boldsymbol{\omega}-\mathbb{E}\left[\boldsymbol{\Phi}^T\boldsymbol{X}^T\boldsymbol{y}\right]\right|\right|_2^2\tag{42}$$

注意到 $\boldsymbol{\Phi}$ 和 $\boldsymbol{\omega}$ 均为线性变换，在实际模型中是非退化的，因此可以认为它们可逆，从而存在另一可逆矩阵 $\boldsymbol{\varphi}$，使

$$\boldsymbol{\Phi}\cdot\boldsymbol{\omega}=(\boldsymbol{\Phi}\cdot\boldsymbol{\varphi})\cdot\left(\boldsymbol{\varphi}^{-1}\cdot\boldsymbol{\omega}\right)\tag{43}$$

因此我们可以固定 $\boldsymbol{\omega}=\boldsymbol{\omega}_0$，对风险函数 $R(\boldsymbol{\Phi},\boldsymbol{\omega})$ 进行化简

$$\begin{aligned}R_{\boldsymbol{\omega}_0}(\boldsymbol{\Phi})&=\sum_{\boldsymbol{X}_i\subset\boldsymbol{X}}J(\boldsymbol{\Phi}\boldsymbol{\omega}_0)+\lambda\cdot\left|\left|\mathbb{E}\left[\boldsymbol{\Phi}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\Phi}\right]\boldsymbol{\omega}-\mathbb{E}\left[\boldsymbol{\Phi}^T\boldsymbol{X}^T\boldsymbol{y}\right]\right|\right|_2^2\\&=\sum_{\boldsymbol{X}_i\subset\boldsymbol{X}}J(\boldsymbol{\Phi}\boldsymbol{\omega}_0)+\lambda\cdot\frac{\partial}{\partial\boldsymbol{\omega}}\left[\frac{1}{2}\left(\boldsymbol{X}\boldsymbol{\Phi}\boldsymbol{\omega}-\boldsymbol{y}\right)^T\left(\boldsymbol{X}\boldsymbol{\Phi}\boldsymbol{\omega}-\boldsymbol{y}\right)\right]_{\boldsymbol{\omega}_0}\\&=\sum_{\boldsymbol{X}_i\subset\boldsymbol{X}}J(\boldsymbol{\Phi}\boldsymbol{\omega}_0)+\lambda\cdot\left|\left|\nabla_{\boldsymbol{\omega}}J(\boldsymbol{\Phi}\boldsymbol{\omega})\right|\right|_{\boldsymbol{\omega}_0}^2\end{aligned}\tag{44}$$

至此，风险函数已经被简化为最终形式。在上述表达式中，惩罚项恰好为损失项的梯度，在算法实现中可以利用自动求导实现便捷计算，大大降低了实现难度。[5, 6]

3.2 实验验证

3.2.1 算法实现

关于真实特征 X 和虚假特征 X'，我们同样建立二元线性回归模型

$$y = \alpha x + \beta x' \tag{45}$$

由于泛化算法可以自动挖掘真实线性关系，我们保留 X 和 X' 直接进行线性回归。[6]

根据前文的讨论，我们采用梯度下降算法求解线性回归参数。损失函数即为前文推导得到的风险函数 $R_{\boldsymbol{\omega}_0}(\boldsymbol{\Phi})$，固定 $\boldsymbol{\omega}_0 = 1.0$，学习率恒定为 10^{-3}，平衡系数恒定为 10^{-5}，不考虑权重衰减。需要指出的是，在小批量梯度下降中，惩罚项梯度模长平方的无偏估计选用

$$\sum_{k=1}^{b} \Big[\nabla_{\boldsymbol{\omega}} L(\boldsymbol{X}_i \boldsymbol{\Phi}\boldsymbol{\omega}, \boldsymbol{y}_i) \cdot \nabla_{\boldsymbol{\omega}} L(\boldsymbol{X}_j \boldsymbol{\Phi}\boldsymbol{\omega}, \boldsymbol{y}_j)\Big]_{\boldsymbol{\omega}=1.0} \tag{46}$$

其中 $(\boldsymbol{X}_i, \boldsymbol{y}_i)$ 和 $(\boldsymbol{X}_j, \boldsymbol{y}_j)$ 为容量均为 b 的两组随机的小批量数据，$L(\boldsymbol{X}, \boldsymbol{y})$ 为具体的损失函数。我们采用与前文相同的分布规则生成数据集

$$\begin{aligned} X &\sim N(0, \sigma^2) \\ Y &= 2X + \varepsilon, \varepsilon \sim N(0, \sigma^2) \\ X' &= Y + \varepsilon', \varepsilon' \sim N(0, 1) \end{aligned} \tag{47}$$

两组数据各包含 10000 个样本，其中第一组数据 $\sigma_1^2 = 1$，第二组数据 $\sigma_2^2 = 0.25$，算法描述如下

算法 2 泛化学习线性回归

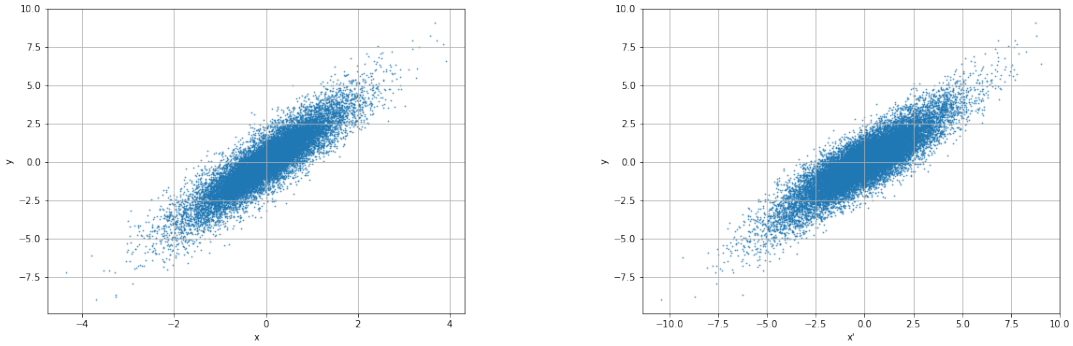
输入： $\boldsymbol{X} \in \mathbb{R}^{n\times 2}$, $\boldsymbol{y} \in \mathbb{R}^n$

输出： $\boldsymbol{\theta} \in \mathbb{R}^2$

$\boldsymbol{\theta} \leftarrow (1,1)^T$
for $k \leftarrow 1$ **to** m **do**
  for $(\boldsymbol{X}_i, \boldsymbol{y}_i) \subset (\boldsymbol{X}, \boldsymbol{y}_i)$ **do**
    SHUFFLE($\boldsymbol{X}_i$, $\boldsymbol{y}_i$)
    $J(\boldsymbol{\theta}) \leftarrow \frac{1}{n}||\boldsymbol{X}_i\boldsymbol{\theta} - \boldsymbol{y}_i||^2$
    $R(\boldsymbol{\theta}) \leftarrow J(\boldsymbol{\theta}) + \lambda \cdot \text{PENALTY}(J(\boldsymbol{\theta}))$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - t \cdot \frac{\partial R(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$
return $\boldsymbol{\theta}$

3.2.2 实验结果

绘制生成数据的散点图，Y 关于 X 和 X' 的分布情况如图所示



(a) Y 关于 X 的分布 (b) Y 关于 X' 的分布

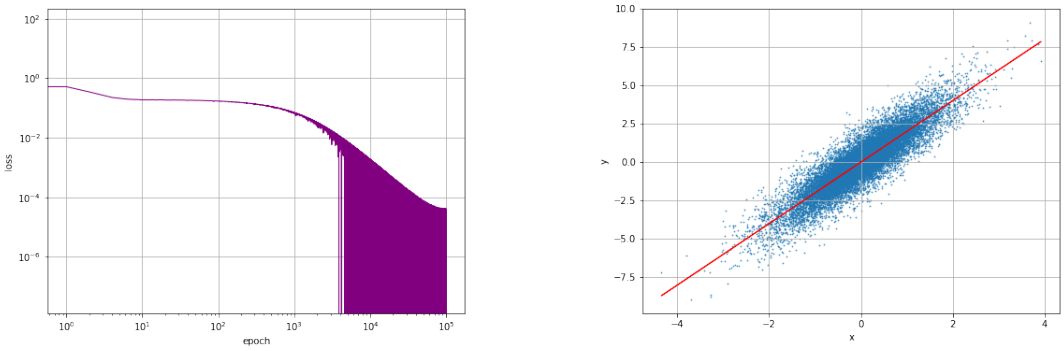
图 6: 生成数据分布情况

与前文相同，$Y \approx 2X \approx X'$，泛化学习算法应当求解出 $y = 2x$ 的真实关系回归结果，而非 $y = x$ 的虚假关系回归结果。

经过足够次数的迭代，模型收敛得到如下线性回归方程

$$y = 2.0048x - 0.0020x' \tag{48}$$

为了反映模型对真实线性关系的回归效果，我们将数据散点和降维直线 $y = \alpha x$ 共同绘制，而忽略 $\beta x'$ 在另一个维度的影响，结果如图所示



(a) 损失函数学习曲线 (b) 真实线性回归结果

图 7: 泛化算法实验结果

可以看到，虚假特征 X' 的权重几乎为 0，而反映真实线性关系的 $\alpha \approx 2$，模型几乎完全摒弃了虚假特征的干扰，完美回归了真实的线性关系。

通过上述实验可知，泛化学习算法在数据不满足独立同分布假设，存在虚假因果关系时，能够从不同的数据分布中充分挖掘信息，排除虚假特征的干扰，得到真实的线性关系。

4 研究总结

4.1 效果分析

经过传统算法和泛化算法的分析和实验，我们将效果对比总结成下表

表 1: 传统算法与泛化算法效果对比

效果	传统算法	泛化算法
回归结果	$\hat{y} = x'$	$\hat{y} = 2x$
正确性	错误	正确
迭代次数	2×10^4	10^5
收敛速度	较快	较慢

可以看到，泛化学习需要大约五倍于传统算法的迭代次数才能达到收敛，因此回归求解速度较慢，但是泛化学习在非独立同分布条件下仍然可以得到正确的线性回归结果。

为了验证泛化学习线性回归的稳定性，我们需要更多组实验的结果支持。我们选择不同的初始斜率 k 构造真实线性关系，而虚假线性关系保持不变，利用上述泛化学习模型进行线性回归，结果如下表所示

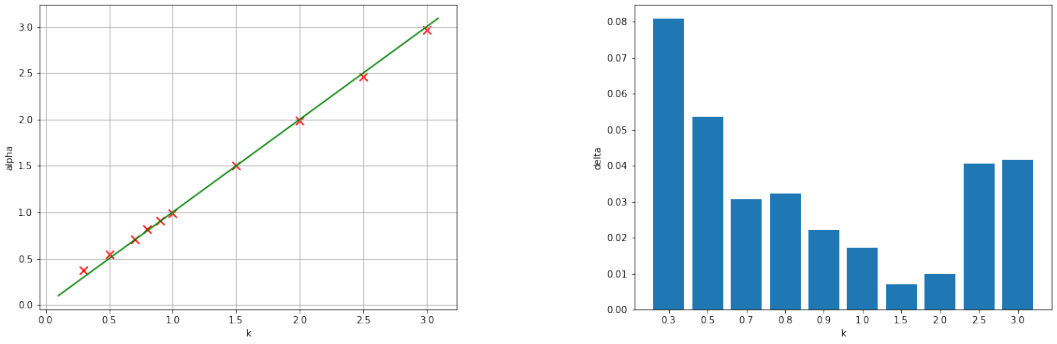
表 2: 泛化学习线性回归结果

实际斜率 k	真实权重 α	虚假权重 β	回归误差 δ
0.3	0.3721	0.0364	0.080767382
0.5	0.5434	0.0314	0.053567901
0.7	0.7112	0.0286	0.030714817
0.8	0.8201	0.0251	0.032156181
0.9	0.9050	0.0214	0.021976351
1.0	0.9928	0.0157	0.017272232
1.5	1.5024	−0.0066	0.007022820
2.0	1.9902	−0.0020	0.010002000
2.5	2.4614	0.0126	0.040604433
3.0	2.9624	0.0181	0.041729726

其中，α 和 β 为线性回归得到的权重，δ 为与标准回归结果的误差

$$\delta = \sqrt{(\alpha - k)^2 + \beta^2} \tag{49}$$

该误差衡量了线性回归与真实情况的偏离程度。我们将 α 和 δ 与 k 的关系绘制如下



(a) 权重系数 α-k 散点图 (b) 回归误差 δ-k 柱状图

图 8: 泛化学习线性回归统计结果

图中绿色直线 $y = x$ 为线性回归结果的标准参考。从图中可以看出，对于 $k \in [0.5, 3.0]$，泛化学习求解的线性回归系数 α 基本与实际情况一致。[1, 2]

表 3: 回归误差 δ 统计结果

均值 μ	**方差** σ^2	**标准差** σ
0.0336	0.0004	0.0210

实验中所有的回归误差 $\delta < 0.1$，从上表可以看出误差均值 $\mu < 0.05$，方差 $\sigma^2 < 0.001$，可见回归结果非常精确。

4.2 研究方法探讨

机器学习领域的研究一直因其糟糕的可解释性（Interpretability）饱受诟病，其原因在于深度学习过度追求优化模型的奇技淫巧，缺少对于基本数学理论的重视。近年来，泛化学习成为机器学习重要的研究领域之一，在这一领域中，独立同分布假设不再作为基本条件，这就要求我们从概率论和数理统计的角度对基本的定理和公式重新进行推导求证，设计新的算法适应泛化学习的要求。

在本文中，我们将线性回归作为基本的研究问题，分别验证传统算法和泛化算法在问题中的表现。在数据构造中，正态分布的生成利用了概率论中的基本方法。在线性回归公式和方法的推导中，最大似然估计和最小二乘法都是数理统计中参数估计的重要方法。在损失函数与梯度下降算法的设计中，拓展运用了凸优化理论。在泛化学习算法的推导中，更加综合运用了概率统计和线性代数中的方法，得到非常优美的结果。在模型效果的分析中，数理统计方法再次发挥了重要作用，散点图、柱状图、数据表都直观地展示了数据的分布情况，让我们的探究有迹可循。

本文的研究非常有价值。一方面，泛化学习本身就是机器学习领域的前沿研究方向之一，具有重要的科学意义。另一方面，本文的探究大量运用了概率统计的知识和方法，并与线性代数、凸优化等跨学科结合，体现了课程要求的科学思维能力。

参考文献

[1] 卫淑芝, 熊德文, 皮玲. 大学数学 概率论与数理统计: 基于案例分析 [M]. 北京: 高等教育出版社, 2020.

[2] 茆诗松, 程依明, 濮晓龙, 等. 概率论与数理统计教程: 第二版 [M]. 北京: 高等教育出版社, 2011.

[3] Bishop C M, Nasrabadi N M. Pattern recognition and machine learning[M]. New York: springer, 2006.

[4] Boyd S, Boyd S P, Vandenberghe L. Convex optimization[M]. Cambridge university press, 2004.

[5] Golub G H, Van Loan C F. Matrix computations[M]. JHU press, 2013.

[6] Arjovsky M, Bottou L, Gulrajani I, et al. Invariant risk minimization[J]. arXiv preprint arXiv:1907.02893, 2019.

[7] Rosenfeld E, Ravikumar P, Risteski A. The risks of invariant risk minimization[J]. arXiv preprint arXiv:2010.05761, 2020.

附录 A 实验代码

A.1 工具函数

```
import torch
import numpy as np
import matplotlib.pyplot as plt

def generate(n=10000, s=1, t=1, k=1):
    x_value = s * torch.randn(n, 1)
    y_value = k * x_value + s * torch.randn(n, 1)
    x_prime = y_value + t * torch.randn(n, 1)
    return torch.cat([x_value, x_prime], dim=1), y_value

def draw_dataset(x_data, y_data, x_label="x", y_label="y",
    filename='./figure/dataset.png'):
    plt.figure(figsize=(8, 6))
    plt.scatter(x_data, y_data, s=0.5, alpha=0.8)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.grid()
    plt.savefig(filename)
    plt.show()

def draw_regression(x_data, y_data, slope, x_label="x", y_label="y",
    filename='./figure/regression.png'):
    plt.figure(figsize=(8, 6))
    plt.scatter(x_data, y_data, s=0.5, alpha=0.8)
    plt.plot(x_data, slope * x_data, c="red", linewidth=1)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.grid()
    plt.savefig(filename)
    plt.show()

def draw_curve(y_data, x_label="epoch", y_label="loss",
    filename='./figure/curve.png'):
    plt.figure(figsize=(8, 6))
    x_data = range(len(y_data))
    x_data, y_data = np.array(x_data), np.array(y_data)
    plt.plot(x_data, y_data, c="purple", linewidth=1)
    plt.loglog()
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.grid()
```

```
    plt.savefig(filename)
    plt.show()
```

A.2 传统算法

```
import torch
from torch import nn, optim
from torch.autograd import grad
import numpy as np
import utils

k = 2.0
sample = [utils.generate(s=1.0, t=0.1, k=k), utils.generate(s=0.1, t=0.1, k=k)]
x1, y1 = sample[0]
x2, y2 = sample[1]
x1, y1 = x1.numpy(), y1.numpy()
x2, y2 = x2.numpy(), y2.numpy()
x_data = np.concatenate([x1, x2], axis=0)
y_data = np.concatenate([y1, y2], axis=0)
utils.draw_dataset(x_data[:, [0]], y_data, x_label="x",
    filename='./figure/traditional_distribution_x_value.png')
utils.draw_dataset(x_data[:, [1]], y_data, x_label="x'",
    filename='./figure/traditional_distribution_x_prime.png')

slope = nn.Parameter(torch.Tensor([[1.0]]))
optimizer = optim.SGD([slope], lr=1e-3)
function = nn.MSELoss(reduction='mean')
for epoch in range(1, 10001):
    loss = 0
    for x, y in sample:
        p = torch.randperm(len(x))
        x, y = x[p], y[p]
        loss += function(x[:, [0]] @ slope, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if epoch % 5000 == 0:
        print('----- Epoch {} -----'.format(epoch))
        print('Loss: {}'.format(loss.item()))
        print('Slope: {}'.format(slope.item()))
k = slope.detach().numpy()[0, 0]
utils.draw_regression(x_data[:, [0]], y_data, k,
    filename='./figure/traditional_regression_real.png')
```

```
slope = nn.Parameter(torch.Tensor([[1.0]]))
optimizer = optim.SGD([slope], lr=1e-3)
function = nn.MSELoss(reduction='mean')
for epoch in range(1, 10001):
    loss = 0
    for x, y in sample:
        p = torch.randperm(len(x))
        x, y = x[p], y[p]
        loss += function(x[:, [1]] @ slope, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if epoch % 5000 == 0:
        print('----- Epoch {} -----'.format(epoch))
        print('Loss: {}'.format(loss.item()))
        print('Slope: {}'.format(slope.item()))
k = slope.detach().numpy()[0, 0]
utils.draw_regression(x_data[:, [0]], y_data, k,
    filename='./figure/traditional_regression_fake.png')

alpha = nn.Parameter(torch.Tensor([[1.0]]))
beta = nn.Parameter(torch.Tensor([[1.0]]))
optimizer = optim.SGD([alpha, beta], lr=1e-3)
function = nn.MSELoss(reduction='mean')
for epoch in range(1, 20001):
    loss = 0
    for x, y in sample:
        p = torch.randperm(len(x))
        x, y = x[p], y[p]
        loss += function(x[:, [0]] @ alpha + x[:, [1]] @ beta, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if epoch % 5000 == 0:
        print('----- Epoch {} -----'.format(epoch))
        print('Loss: {}'.format(loss.item()))
        print('Alpha: {}'.format(alpha.item()))
        print('Beta: {}'.format(beta.item()))
k = alpha.detach().numpy()[0, 0]
utils.draw_regression(x_data[:, [0]], y_data, k,
    filename='./figure/traditional_regression_mixed.png')
```

A.3 泛化算法

```
import torch
from torch import nn, optim
from torch.autograd import grad
import numpy as np
import utils

k = 2.0
sample = [utils.generate(s=1.0, k=k), utils.generate(s=0.5, k=k)]
x1, y1 = sample[0]
x2, y2 = sample[1]
x1, y1 = x1.numpy(), y1.numpy()
x2, y2 = x2.numpy(), y2.numpy()
x_data = np.concatenate([x1, x2], axis=0)
y_data = np.concatenate([y1, y2], axis=0)
utils.draw_dataset(x_data[:, [0]], y_data, x_label="x",
    filename='./figure/generalized_distribution_x_value.png')
utils.draw_dataset(x_data[:, [1]], y_data, x_label="x'",
    filename='./figure/generalized_distribution_x_prime.png')

def evaluate(loss, omega):
    g1 = grad(loss[0::2].mean(), omega, create_graph=True)[0]
    g2 = grad(loss[1::2].mean(), omega, create_graph=True)[0]
    return (g1 * g2).sum()

theta = nn.Parameter(torch.ones(2, 1))
omega = nn.Parameter(torch.Tensor([1.0]))
optimizer = optim.SGD([theta], lr=1e-3)
function = nn.MSELoss(reduction='none')
record = []
for epoch in range(1, 100001):
    error = 0
    penalty = 0
    for x, y in sample:
        p = torch.randperm(len(x))
        x, y = x[p], y[p]
        loss = function(x @ theta * omega, y)
        error += loss.mean()
        penalty += evaluate(loss, omega)
    optimizer.zero_grad()
    (1e-5 * error + penalty).backward()
    optimizer.step()
    record.append((1e-5 * error + penalty).item())
    if epoch % 5000 == 0:
        print('----- Epoch {} -----'.format(epoch))
        print('Error: {}'.format(error.item()))
```

```
        print('Penalty: {}'.format(penalty.item()))
        print(theta)
k = theta.detach().numpy()[0, 0]
utils.draw_regression(x_data[:, [0]], y_data, k,
    filename='./figure/generalized_regression.png')
utils.draw_curve(record, filename='./figure/generalized_curve.png')
```

A.4 效果测试

```
import torch
from torch import nn, optim
from torch.autograd import grad

def generate(n=10000, s=1, t=1, k=1):
    x_value = s * torch.randn(n, 1)
    y_value = k * x_value + s * torch.randn(n, 1)
    x_prime = y_value + t * torch.randn(n, 1)
    return torch.cat([x_value, x_prime], dim=1), y_value

def evaluate(loss, omega):
    g1 = grad(loss[0::2].mean(), omega, create_graph=True)[0]
    g2 = grad(loss[1::2].mean(), omega, create_graph=True)[0]
    return (g1 * g2).sum()

ks = [0.3, 0.5, 0.7, 0.8, 0.9, 1.0, 1.5, 2.0, 2.5, 3.0]
for k in ks:
    print("\n---------- k = {} ----------".format(k))
    sample = [generate(s=1.0, k=k), generate(s=0.1, k=k)]
    theta = nn.Parameter(torch.ones(2, 1))
    omega = nn.Parameter(torch.Tensor([1.0]))
    optimizer = optim.SGD([theta], lr=1e-3)
    function = nn.MSELoss(reduction='none')
    record = []
    for epoch in range(1, 100001):
        error = 0
        penalty = 0
        for x, y in sample:
            p = torch.randperm(len(x))
            x, y = x[p], y[p]
            loss = function(x @ theta * omega, y)
            error += loss.mean()
            penalty += evaluate(loss, omega)
        optimizer.zero_grad()
        (1e-5 * error + penalty).backward()
```

```
        optimizer.step()
        record.append((1e-5 * error + penalty).item())
        if epoch % 5000 == 0:
            print('----- Epoch {} -----'.format(epoch))
            print('Error: {}'.format(error.item()))
            print('Penalty: {}'.format(penalty.item()))
            print(theta)
```

A.5 数据分析

```
import numpy as np
import matplotlib.pyplot as plt

k = np.array([
    0.3, 0.5, 0.7, 0.8, 0.9,
    1.0, 1.5, 2.0, 2.5, 3.0
])
alpha = np.array([
    0.3721, 0.5434, 0.7112, 0.8201, 0.9050,
    0.9928, 1.5024, 1.9902, 2.4614, 2.9624
])
beta = np.array([
    0.0364, 0.0314, 0.0286, 0.0251, 0.0214,
    0.0157, -0.0066, -0.0020, 0.0126, 0.0181
])
delta = np.sqrt(np.square(k - alpha) + np.square(beta))
mu = np.mean(delta)
sigma2 = np.var(delta)
sigma = np.sqrt(sigma2)
print("delta:", delta)
print("mu:", mu)
print("sigma^2:", sigma2)
print("sigma:", sigma)

plt.figure(figsize=(8, 6))
std_x = np.arange(0.1, 3.1, 0.01)
std_y = std_x
plt.plot(std_x, std_y, c="green")
plt.scatter(k, alpha, s=75, c="red", marker="x")
plt.grid()
plt.xlabel("k")
plt.ylabel("alpha")
plt.savefig("./figure/statistic_weight.png")
```

附录 B 运行输出

B.1 传统算法

B.1.1 真实特征

```
----- Epoch 5000 -----
Loss: 1.0244024991989136
Slope: 1.9956578016281128
----- Epoch 10000 -----
Loss: 1.0244026184082031
Slope: 1.995678186416626
```

B.1.2 虚假特征

```
----- Epoch 5000 -----
Loss: 0.01996424049139023
Slope: 0.9956846237182617
----- Epoch 10000 -----
Loss: 0.019964244216680527
Slope: 0.9956846237182617
```

B.1.3 混合特征

```
----- Epoch 5000 -----
Loss: 0.023706747218966484
Alpha: 0.17271681129932404
Beta: 0.9251658916473389
----- Epoch 10000 -----
Loss: 0.01983446441590786
Alpha: 0.05767692252993584
Beta: 0.9724671244621277
----- Epoch 15000 -----
Loss: 0.01972406730055809
Alpha: 0.03825797140598297
Beta: 0.9804511070251465
----- Epoch 20000 -----
Loss: 0.019720911979675293
Alpha: 0.034980714321136475
Beta: 0.9818002581596375
```

B.2 泛化算法

```
----- Epoch 5000 -----
```

```
Error: 0.8509488701820374
Penalty: 0.007319848518818617
Parameter containing:
tensor([[1.5826],
        [0.2368]], requires_grad=True)
----- Epoch 10000 -----
Error: 0.9402514696121216
Penalty: 0.0005529667250812054
Parameter containing:
tensor([[1.7274],
        [0.1626]], requires_grad=True)
----- Epoch 15000 -----
Error: 0.9959583282470703
Penalty: 0.0005683571216650307
Parameter containing:
tensor([[1.7932],
        [0.1267]], requires_grad=True)
----- Epoch 20000 -----
Error: 1.0348505973815918
Penalty: -0.0003279283409938216
Parameter containing:
tensor([[1.8328],
        [0.1043]], requires_grad=True)
----- Epoch 25000 -----
Error: 1.0643324851989746
Penalty: -0.0013250481570139527
Parameter containing:
tensor([[1.8601],
        [0.0884]], requires_grad=True)
----- Epoch 30000 -----
Error: 1.088323950767517
Penalty: -0.00041986300493590534
Parameter containing:
tensor([[1.8809],
        [0.0761]], requires_grad=True)
----- Epoch 35000 -----
Error: 1.1084226369857788
Penalty: 2.416716597508639e-06
Parameter containing:
tensor([[1.8974],
        [0.0662]], requires_grad=True)
----- Epoch 40000 -----
Error: 1.126463770866394
Penalty: -0.0003694269107654691
Parameter containing:
```

```
tensor([[1.9116],
        [0.0575]], requires_grad=True)
----- Epoch 45000 -----
Error: 1.1422295570373535
Penalty: -0.00012091264215996489
Parameter containing:
tensor([[1.9237],
        [0.0502]], requires_grad=True)
----- Epoch 50000 -----
Error: 1.1566444635391235
Penalty: -0.0007210643962025642
Parameter containing:
tensor([[1.9342],
        [0.0436]], requires_grad=True)
----- Epoch 55000 -----
Error: 1.1698400974273682
Penalty: -0.005578203592449427
Parameter containing:
tensor([[1.9437],
        [0.0377]], requires_grad=True)
----- Epoch 60000 -----
Error: 1.182608962059021
Penalty: -0.001036302768625319
Parameter containing:
tensor([[1.9526],
        [0.0320]], requires_grad=True)
----- Epoch 65000 -----
Error: 1.194344401359558
Penalty: -0.0015861056745052338
Parameter containing:
tensor([[1.9606],
        [0.0269]], requires_grad=True)
----- Epoch 70000 -----
Error: 1.2054845094680786
Penalty: 2.2196682039066218e-05
Parameter containing:
tensor([[1.9681],
        [0.0222]], requires_grad=True)
----- Epoch 75000 -----
Error: 1.2162278890609741
Penalty: -0.00011362621444277465
Parameter containing:
tensor([[1.9750],
        [0.0177]], requires_grad=True)
----- Epoch 80000 -----
```

```
Error: 1.2263251543045044
Penalty: -0.0009810078190639615
Parameter containing:
tensor([[1.9815],
        [0.0135]], requires_grad=True)
----- Epoch 85000 -----
Error: 1.2362619638442993
Penalty: -0.002543547423556447
Parameter containing:
tensor([[1.9877],
        [0.0094]], requires_grad=True)
----- Epoch 90000 -----
Error: 1.245991587638855
Penalty: -0.0008340947097167373
Parameter containing:
tensor([[1.9937],
        [0.0054]], requires_grad=True)
----- Epoch 95000 -----
Error: 1.2556943893432617
Penalty: -0.00012431324284989387
Parameter containing:
tensor([[1.9996e+00],
        [1.5151e-03]], requires_grad=True)
----- Epoch 100000 -----
Error: 1.2645938396453857
Penalty: -0.0003699126245919615
Parameter containing:
tensor([[ 2.0048],
        [-0.0020]], requires_grad=True)
```

B.3 效果测试

```
---------- k = 0.3 ----------
Error: 0.9352296590805054
Penalty: 1.8052724044537172e-06
Parameter containing:
tensor([[0.3721],
        [0.0364]], requires_grad=True)
---------- k = 0.5 ----------
Error: 0.9494261145591736
Penalty: -5.543452061829157e-06
Parameter containing:
tensor([[0.5434],
        [0.0314]], requires_grad=True)
---------- k = 0.7 ----------
```

```
Error: 0.9772871136665344
Penalty: -0.00020351592684164643
Parameter containing:
tensor([[0.7112],
        [0.0286]], requires_grad=True)
---------- k = 0.8 ----------
Error: 0.9810543656349182
Penalty: -0.0008637678110972047
Parameter containing:
tensor([[0.8201],
        [0.0251]], requires_grad=True)
---------- k = 0.9 ----------
Error: 0.9486773610115051
Penalty: -0.0008569502970203757
Parameter containing:
tensor([[0.9050],
        [0.0214]], requires_grad=True)
---------- k = 1.0 ----------
Error: 0.9571946859359741
Penalty: -0.0002945401065517217
Parameter containing:
tensor([[0.9928],
        [0.0157]], requires_grad=True)
---------- k = 1.5 ----------
Error: 1.0061291456222534
Penalty: -0.0008013449260033667
Parameter containing:
tensor([[ 1.5024],
        [-0.0066]], requires_grad=True)
---------- k = 2.0 ----------
Error: 1.0208115577697754
Penalty: -0.0021623382344841957
Parameter containing:
tensor([[ 1.9902e+00],
        [-1.9645e-03]], requires_grad=True)
---------- k = 2.5 ----------
Error: 0.9971625208854675
Penalty: -0.0008912317571230233
Parameter containing:
tensor([[2.4614],
        [0.0126]], requires_grad=True)
---------- k = 3.0 ----------
Error: 0.9859713912010193
Penalty: -0.0011869616573676467
Parameter containing:
```

```
tensor([[2.9624],
        [0.0181]], requires_grad=True)
```

B.4 数据分析

```
delta: [0.08076738 0.0535679 0.03071482 0.03215618 0.02197635 0.01727223
 0.00702282 0.010002 0.04060443 0.04172973]
mu: 0.03358138430967583
sigma^2: 0.0004413956278458581
sigma: 0.021009417598921158
```