LinearRegression线性回归

方程建立

对于因变量y，有n维向量

$$x = [x_1, x_2, \dots, x_n]$$

这样一系列影响因素影响。
此外线性模型还有一个n维的权重和一个标量偏差

$$w = [w_1, w_2, \dots, w_n]^T,\ \ b$$

输出是

$$y =< w, x > +b$$

其中<w,x>是w与x的内积

对于预测值和真实值的偏差，引入损失函数

$$e(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

拟合参数计算

线性回归的拟合过程可以看做是对单层神经网络的训练，输入为向量x，权重为向量w，输出为y。
对于已有数据进行拟合相当于训练数据。
对于损失可以记作：

$$e(X, y, w, b) = \frac{1}{2n}||y - Xw - b||^2$$

最小化损失函数来学习参数权重，将偏差加入权重

$$X \leftarrow [X, 1],\ \ w \leftarrow [w, b]^T$$

经过求偏导计算得到最优解满足

$$w^* = (X^T X)^{-1} X y$$

梯度下降

- 挑选初始参数值w_0
- 重复迭代参数$t = 1, 2, 3, \ldots$
- $w_t = w_{t-1} - \eta \frac{\partial e}{\partial w_{t-1}}$
- 沿着梯度方向将增加损失函数值
- 学习率η：步长的超参数，需要选取合适
- 学习率，不能太大不能太小，学习率太大，可能会导致训练不收敛或者震荡，学习率太小，可能导致收敛过慢或者无法跳出局部最优解
  小批量随机梯度下降
  损失是在所有样本损失的平均
  我们可以随机采样b个样本损失均值来近似损失，b为批量大小。

常用的损失函数

- 均方损失L2 Loss：$l(y, y') = \frac{1}{2}(y - y')^2$
- 绝对值损失L1 Loss：$l(y, y') = |y - y'|$
- Huber's Robust Loss：

$$l(y, y') = \begin{cases} |y - y'| - \frac{1}{2}, & \text{if } |y - y'| > 1 \\ \frac{1}{2}(y - y')^2, & \text{otherwise} \end{cases}$$

线性回归手动实现

```
# 导包
import random
import torch
import matplotlib.pyplot as plt
```

```python
# 1. 构造人造数据集
def synthetic_data(w, b, num_examples):
    """构造人造数据集 y = Xw + b + e"""
    X = torch.normal(0, 1, (num_examples, len(w)))  # 自变量
    y = torch.matmul(X, w) + b  # 回归方程
    y += torch.normal(0, 0.01, y.shape)  # 随机误差
    return X, y.reshape((-1, 1))  # 返回特征 X 和标签 y（列向量）

# 设定真实的 w 和 b
true_w = torch.tensor([2, -3.4])
true_b = 4.2
features, labels = synthetic_data(true_w, true_b, 1000)

# 2. 可视化数据（x1 对 y 的散点图）
plt.scatter(features[:, 1].detach().numpy(),
labels.detach().numpy(), alpha=0.5)
plt.xlabel('x1')
plt.ylabel('y')
plt.title('y vs x1')
plt.show()

# 3. 小批量数据迭代器
def data_iter(batch_size, features, labels):
    """小批量数据迭代器"""
    num_examples = len(features)
    indices = list(range(num_examples))  # 生成 0 到 999 的索引列表
    random.shuffle(indices)  # 打乱索引顺序
    for i in range(0, num_examples, batch_size):
        batch_indices =
torch.tensor(indices[i:min(i+batch_size,
num_examples)])  # 当前批次的索引
        yield features[batch_indices],
labels[batch_indices]  # 返回当前批次的特征和标签
```

```python
# 4. 初始化模型参数
w = torch.normal(0, 0.01, size=(2, 1),
requires_grad=True)  # 初始化权重
b = torch.zeros(1, requires_grad=True)  # 初始化偏置

# 5. 线性回归模型
def linreg(X, w, b):
    """线性回归模型: y_hat = Xw + b"""
    return torch.matmul(X, w) + b

# 6. 损失函数（平方损失）
def squared_loss(y_hat, y):
    """平方损失函数: L = 1/2 * Σ(ŷ - y)^2"""
    return (y_hat - y.reshape(y_hat.shape)) ** 2 / 2

# 7. 随机梯度下降（SGD）
def sgd(params, lr, batch_size):
    """随机梯度下降更新参数"""
    with torch.no_grad():
        for param in params:
            param -= lr * param.grad / batch_size  # 更
新参数
            param.grad.zero_()  # 清零梯度

# 8. 训练过程
lr = 0.03  # 学习率
num_epochs = 3  # 迭代次数
batch_size = 10  # 批次大小
net = linreg  # 选择线性回归模型
loss = squared_loss  # 选择平方损失函数

# 训练过程
for epoch in range(num_epochs):
    for X, y in data_iter(batch_size, features,
labels):
```

```
            # 计算损失和梯度
            l = loss(net(X, w, b), y)
            l.sum().backward()  # 反向传播计算梯度
            sgd([w, b], lr, batch_size)  # 使用SGD更新参数

        # 计算整个训练集上的损失
        with torch.no_grad():
            train_l = loss(net(features, w, b), labels)
            print(f'epoch {epoch+1}, loss
{float(train_l.mean()):f}')

# 9. 打印估计误差
print(f'w的估计误差: {true_w - w.reshape(true_w.shape)}')
print(f'b的估计误差: {true_b - b}')
```

线性回归简洁实现

```
# 导入必要的库
import numpy as np
import torch
from torch.utils import data
from torch import nn

# 1. 构造人造数据集：y = Xw + b + e
def synthetic_data(w, b, num_examples):
    """构造人造数据集 y = Xw + b + e"""
    X = torch.normal(0, 1, (num_examples, len(w)))  #
自变量X，正态分布
    y = torch.matmul(X, w) + b  # 回归方程：y = Xw + b
    y += torch.normal(0, 0.01, y.shape)  # 添加随机误差
    return X, y.reshape((-1, 1))  # 返回特征 X 和标签 y
（列向量）

# 2. 设置真实的w和b
```

```python
true_w = torch.tensor([2, -3.4])  # 真实的w
true_b = 4.2  # 真实的b

# 3. 生成数据集
features, labels = synthetic_data(true_w, true_b, 1000)

# 4. 数据加载器函数
def load_array(data_arrays, batch_size, is_train=True):
    """小批量数据迭代器"""
    dataset = data.TensorDataset(*data_arrays)  # 创建TensorDataset
    return data.DataLoader(dataset, batch_size, shuffle=is_train)  # 使用DataLoader进行批处理

# 5. 设置批量大小并加载数据
batch_size = 10
data_iter = load_array((features, labels), batch_size)

# 6. 打印一个batch的数据
next(iter(data_iter))

# 7. 定义模型
net = nn.Sequential(nn.Linear(2, 1))  # 线性回归模型，2个输入，1个输出
net[0].weight.data.normal_(0, 0.01)  # 初始化权重
net[0].bias.data.fill_(0)  # 初始化偏置
loss = nn.MSELoss()  # 使用PyTorch的均方误差损失函数

# 8. 定义优化器
trainer = torch.optim.SGD(net.parameters(), lr=0.03)  # 随机梯度下降优化器

# 9. 训练过程
num_epochs = 3  # 训练轮数
for epoch in range(num_epochs):
    for X, y in data_iter:  # 每个batch的数据
```

```
        l = loss(net(X), y)  # 计算损失
        trainer.zero_grad()  # 清空之前的梯度
        l.backward()  # 反向传播
        trainer.step()  # 更新参数
    l = loss(net(features), labels)  # 计算当前训练集的损失
    print(f'epoch {epoch+1}, loss {l.item():.6f}')  # 输出当前epoch的损失值

# 10. 打印估计误差
print(f'w的估计误差: {true_w - net[0].weight.data.reshape(true_w.shape)}')
print(f'b的估计误差: {true_b - net[0].bias.data}')
```