通网综合专题大作业—机器学习-python 代码学习及运行

姓名：贾生萍　　班级：通信 1707　　学号：17211390

一、　相关背景知识

回归是机器学习中的一个基础的概念，即从有限的现实事件中提取一系列的特征，通过发现这些特征之间的关系，寻找规律，来通过已有的特征来预测或判断一个相关的结果，对应数学概念来说，即建立一个数学模型（建模），并对一些已有数据提取其特征转换为自变量，将这一系列自变量输入到模型之中（输入），经过模型计算得到因变量（输出），通过把计算得到的因变量与已有的真实因变量对比（使用损失函数），应用一些优化方法来优化模型（训练），使用训练过的模型来对一些已有的自变量去预测其结果（预测）。

数学中的回归方法很多，各类回归方法虽然在一些细节方面有所差异，但本质上却是比较一致的，不同的回归方法的提出，主要基于不同的应用场景，当需要解释的现实规律不同时，需要找出不同的模型来进行适配，而线性回归与逻辑回归即为比较典型的两种类型。

二、　原理及建模过程

1. 线性回归

一种当自变量与因变量之间主要呈现线性关系的经典模型；主要包括模型部分（即线性关系）、数据集、损失函数、优化函数这几个基本要素。

- 模型部分

线性回归模型的输出与各个输入之间是线性关系，构建线性模型即可

- 数据集

通常会收集一系列的真实数据，并在这个数据上面寻找模型参数来使模型的预测值与真实值的误差最小。这个数据集被称为训练数据集（training data set）或训练集（training set），一条监测记录被称为一个样本（sample），其输出值叫作标签（label），用来预测标签的因素叫作特征（feature）。

- 损失函数

在模型训练中，我们需要衡量价格预测值与真实值之间的误差。通常会选取一个　　非负数作为误差，且数值越小表示误差越小。一般采用均方差函数（MSE）

- 优化函数

当模型和损失函数形式较为简单时，上面的误差最小化问题的解可以直接用公式表达出来。这类解叫作解析解。然而，大多数深度学习模型并没有解析解，即模型特别复杂，只能通过优化算法有限次迭代模型参数来尽可能降低损失函数的值。这类解叫作数值解。

在求数值解的优化算法中，一般采用小批量随机梯度下降的方法：先选取一组模型参数的初始值，一般是随机选取；接下来对参数进行多次迭代，使每次迭代都可能降低损失函数的值。在每次迭代中，先随机均匀采样一个由固定数目训练数据样本所组成的小批量，然后求小批量中数据样本的平均损失有关模型参数的导数（梯度），最后用此结果与预先设定的一个正数的乘积作为模型参数在本次迭代的减小量。一般步骤如下：

- 初始化模型参数，一般来说使用随机初始化；

➢ 在数据上迭代多次，通过在负梯度方向移动参数来更新每个参数。

2．**逻辑回归**

常见逻辑回归解决的问题中，因变量一般是 true or false 的关系，其结果是一个离散的值，一般用于二分类。它主要是在线性回归的基础上，将结果输入到 softmax 函数（Softmax 函数计算事件超过’n’个不同事件的概率分布）中去，即将该连续结果映射到 0-1 的区间中，用于多分类模型，输出的结果是分类为 True 的概率，并且目标类别的概率值会很大。

与线性建模的过程基本类似，但对于逻辑回归的损失函数来说，由于对输出加入了 softmax 激活函数，如果采用均方差作为损失函数，当输出接近 0 或者 1 时，会出现梯度消失的问题，所以需要采用交叉熵来避免这个问题。之后再采用梯度下降法来优化参数，使损失函数逐步趋于最优解，以此来进行模型的优化，使训练数据的标签值与预测出来的值之间的误差最小化。

三、 代码实现

1. 线性回归

在此给出代码的重要部分截图，完整代码及注释将附在本文档最后部分；

- 数据集

```
# Toy dataset
x_train = np.array([[3.3], [4.4], [5.5], [6.71], [6.93], [4.168],##生成训练数组x，fl
                    [9.779], [6.182], [7.59], [2.167], [7.042],
                    [10.791], [5.313], [7.997], [3.1]], dtype=np.float32)
y_train = np.array([[1.7], [2.76], [2.09], [3.19], [1.694], [1.573],##生成训练数组y
                    [3.366], [2.596], [2.53], [1.221], [2.827],
                    [3.465], [1.65], [2.904], [1.3]], dtype=np.float32)
```

- 线性模型

```
# Linear regression model

model = nn.Linear(input_size, output_size)
```

- 损失函数和优化算法

```
# Loss and optimizer; 损失和优化器

criterion = nn.MSELoss()##损失函数

optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)##SGD优化算法
```

2. 逻辑回归

- 训练样本和测试样本

```
# MNIST dataset (images and labels)样本和标签
train_dataset = torchvision.datasets.MNIST(root='../../data',train=True,transform=transforms.ToTensor(),download=True)
test_dataset = torchvision.datasets.MNIST(root='../../data',train=False,transform=transforms.ToTensor())
# Data loader (input pipeline),数据的下载
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,batch_size=batch_size,shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,batch_size=batch_size,shuffle=False)
```

- 逻辑回归（利用线性拟合，从大量的数据中找规律，实现逻辑回归）

```
# Logistic regression model逻辑回归模型
model = nn.Linear(input_size, num_classes)
```

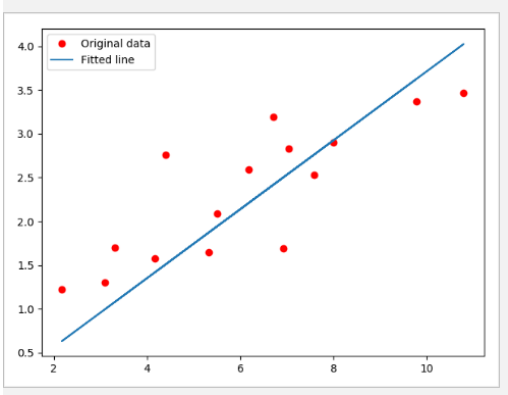
- 损失函数及优化（CrossEntropyLoss 中包括了 softmax 分类和损失函数交叉熵）

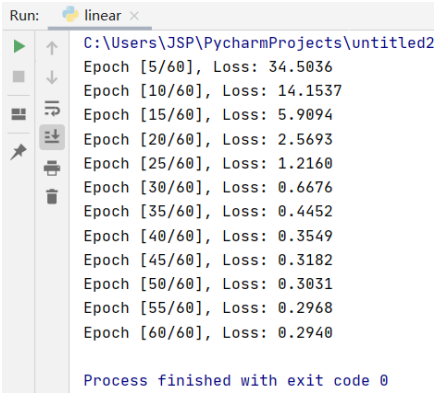
```
# Loss and optimizer
# nn.CrossEntropyLoss() computes softmax internally
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

- 训练和测试模型(测试阶段不需要计算梯度),具体代码及注释在代码附录中给出。

四、 运行结果

1. 线性回归





从结果可以看出，损耗函数值在变小（即迭代收敛），也就是预测值和真实值之间的误差在每一次的迭代之后在不断的变小，这样的迭代优化算法之后，可以得到误差小的线性回归模型参数。

2. 逻辑回归

```
Epoch [1/5], Step [100/600], Loss: 2.2139
Epoch [1/5], Step [200/600], Loss: 2.0953
Epoch [1/5], Step [300/600], Loss: 2.0040
Epoch [1/5], Step [400/600], Loss: 1.9098
Epoch [1/5], Step [500/600], Loss: 1.8025
Epoch [1/5], Step [600/600], Loss: 1.7551
Epoch [2/5], Step [100/600], Loss: 1.7344
Epoch [2/5], Step [200/600], Loss: 1.7145
Epoch [2/5], Step [300/600], Loss: 1.5876
Epoch [2/5], Step [400/600], Loss: 1.5805
Epoch [2/5], Step [500/600], Loss: 1.5631
Epoch [2/5], Step [600/600], Loss: 1.4614
Epoch [3/5], Step [100/600], Loss: 1.3950
Epoch [3/5], Step [200/600], Loss: 1.2307
Epoch [3/5], Step [300/600], Loss: 1.3738
Epoch [3/5], Step [400/600], Loss: 1.3444
Epoch [3/5], Step [500/600], Loss: 1.2624
Epoch [3/5], Step [600/600], Loss: 1.2643
Epoch [4/5], Step [100/600], Loss: 1.1968
Epoch [4/5], Step [200/600], Loss: 1.0495
Epoch [4/5], Step [300/600], Loss: 1.1068
Epoch [4/5], Step [400/600], Loss: 1.1456
Epoch [4/5], Step [500/600], Loss: 1.1221
Epoch [4/5], Step [600/600], Loss: 1.1684
Epoch [5/5], Step [100/600], Loss: 1.0608
Epoch [5/5], Step [200/600], Loss: 1.0459
Epoch [5/5], Step [300/600], Loss: 0.9784
Epoch [5/5], Step [400/600], Loss: 1.0480
Epoch [5/5], Step [500/600], Loss: 1.0270
Epoch [5/5], Step [600/600], Loss: 0.9504
Accuracy of the model on the 10000 test images: 83 %
```

从结果来看，损失函数在不断减小，及不断优化；且根据逻辑回归的原理可知，输出的结果是分类为 True 的概率，从结果来看，概率为 83%，概率值高，一般为目标类别。

五、 心得体会

这是我第一次接触 python，对它的学习是通过和同学的沟通和在网上查阅资料，于是渐渐对它有了了解；同时通过学习也基本了解了线性和逻辑回归的原理及建模过程，对神经网络和深度学习等新鲜的字眼有了一定的认识。但我知道，我的了解肯定还不足以，学习的还只是一些皮毛，但通过这次大作业收获到了很多新的知识，对我来说还是很有帮助的。

【附：源代码及解释】

1. 线性回归模型：

```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
# Hyper-parameters
input_size = 1
output_size = 1
num_epochs = 60#训练重复的次数
learning_rate = 0.001#学习率：η代表在每次优化中，能够学习的步长的大小
# Toy dataset，数据集
#输入的值 x，以此得到根据模型计算得到的因变量
x_train = np.array([[3.3], [4.4], [5.5], [6.71], [6.93], [4.168], [9.779], [6.182], [7.59], [2.167], [7.042],
                    [10.791], [5.313], [7.997], [3.1]], dtype=np.float32)
#生成训练数组 y，真实因变量
y_train = np.array([[1.7], [2.76], [2.09], [3.19], [1.694], [1.573],[3.366], [2.596], [2.53], [1.221], [2.827],
                    [3.465], [1.65], [2.904], [1.3]], dtype=np.float32)
# Linear regression model，线性模型
model = nn.Linear(input_size, output_size)
# Loss and optimizer；损失和优化器
criterion = nn.MSELoss()##损失函数
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)##SGD 优化算法
# Train the model 训练模型，即优化模型
for epoch in range(num_epochs):
    # Convert numpy arrays to torch tensors 将数组转换成张量
    inputs = torch.from_numpy(x_train)
    targets = torch.from_numpy(y_train)
    # Forward pass 前向传播计算 loss
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    # Backward and optimize
    optimizer.zero_grad()#梯度清零，因为梯度会累加，后一次会在前一次的基础上累加，如果不进行梯度清零，梯度就会无限下降
    loss.backward()#optimizer 基于反向梯度更新参数空间
    optimizer.step()
    if (epoch + 1) % 5 == 0:
        print('Epoch [{}/{}], Loss: {:.4f}'.format(epoch + 1, num_epochs, loss.item()))
# Plot the graph 画图
predicted = model(torch.from_numpy(x_train)).detach().numpy()
plt.plot(x_train, y_train, 'ro', label='Original data')
plt.plot(x_train, predicted, label='Fitted line')
plt.legend()
```

```
plt.show()
torch.save(model.state_dict(), 'model.ckpt') # Save the model checkpoint
```

2. 逻辑回归模型

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
# Hyper-parameters
input_size = 28 * 28    # 784
num_classes = 10##标签类别总数
num_epochs = 5##重复训练的次数
batch_size = 100##单次训练用的样本数，小批量计算中的批量大小(batch size)
learning_rate = 0.001#学习率，能够学习的步长的大小
# MNIST dataset (images and labels)样本和标签
#创建数据集 training.pt；从 internet 下载数据集并将其放在根目录中 root；并进行函数转换，将 pil 图片转换为张量
train_dataset = torchvision.datasets.MNIST(root='../../data', train=True,
                                           transform=transforms.ToTensor(), download=True)
##创建数据集 test.pt；
test_dataset = torchvision.datasets.MNIST(root='../../data',    train=False,
                                          transform=transforms.ToTensor())
# Data loader (input pipeline),数据的下载
#定义一个新的迭代器，实现 batch 读取；对 train-dataset 数据集实现批量抽取，shuffle（设置为真不会按数据顺序输出，一般在训练数据中使用）
train_loader = torch.utils.data.DataLoader(dataset=train_datas，  batch_size=batch_size,
                                           shuffle=True)
##对 train-dataset 数据集实现批量抽取，shuffle 设置为 False 依次取出数据
 test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size,
                                           shuffle=False)
# Logistic regression model 逻辑回归模型，利用线性拟合，从大量数据中找规律
model = nn.Linear(input_size, num_classes)
# Loss and optimizer，损失函数和优化，损失函数交叉熵，SGD 优化算法
# nn.CrossEntropyLoss() computes softmax internally
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
# Train the model，训练模型
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Reshape images to (batch_size, input_size)对图像进行切割
        #将数据拆分重塑为模型需要的大小
        images = images.reshape(-1, input_size)
        # Forward pass，前向传播计算模型的 loss
        outputs = model(images)
```

```
            loss = criterion(outputs, labels)
            # Backward and optimize，后向传播，更新模型的参数
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()#迭代更新
            if (i + 1) % 100 == 0:#每 100 个 batch 打印一次数据
                print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                      .format(epoch + 1, num_epochs, i + 1, total_step, loss.item()))
# Test the model，测试阶段，不用计算梯度
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.reshape(-1, input_size)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum()
    print('Accuracy of the model on the 10000 test images: {} %'.format(100 * correct /
total))
# Save the model checkpoint
torch.save(model.state_dict(), 'model.ckpt')
```