**Project6-Linear**

一、运行环境

操作系统：Windows11

Python版本：3.7

第三方库：numpy 1.21.6

二、过程分析

（1）线性回归

令表示特征矩阵（为样本个数，为特征维数），表示年龄矩阵，权重是一个维列向量，偏置是一个标量。则对于第号样本，估计年龄为：

①解析解：

令参数矩阵，自变量矩阵，其中，是元素全为1的矩阵，则最小二乘法的目标是使得的值最小。而

（由于与均为标量，故二者相等）

于是有

由矩阵的链式求导法则：

故

令上式为0，可得

②梯度下降法：

定义损失函数为：

则梯度为：

根据这个梯度，结合步长可以更新权重和偏置：

初始的权重和偏置可以由高斯噪声生成器来生成。

③随机梯度下降法：

每次从全部样本中随机抽取B个样本，然后利用这B个样本构建一个损失函数并计算梯度，最后更新权重和偏置。除了样本个数不是全部样本而是随机抽取的外，其余更新迭代步骤与梯度下降法相同。

（2）线性分类

令表示特征矩阵（为样本个数，为特征维数），表示分类矩阵（元素取值范围为-1和1），权重是一个维列向量，偏置是一个标量。则对于第号样本，将其分类为：

定义单个样本的损失函数为：

则单个样本的梯度为：

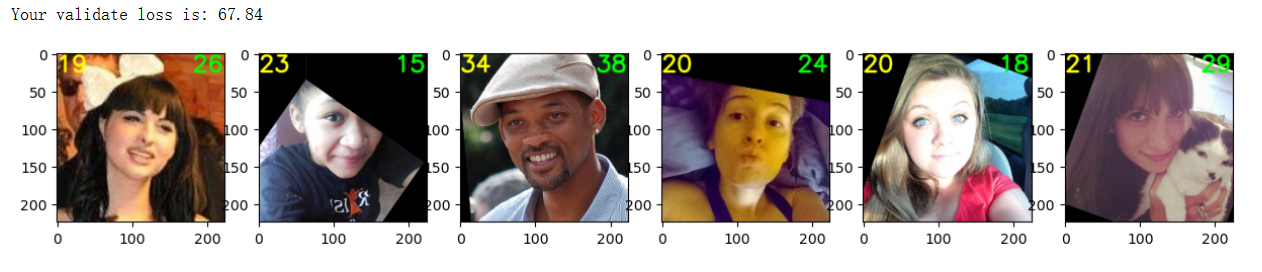
根据这个梯度，结合步长可以更新权重和偏置：

初始的权重和偏置可以由高斯噪声生成器来生成。

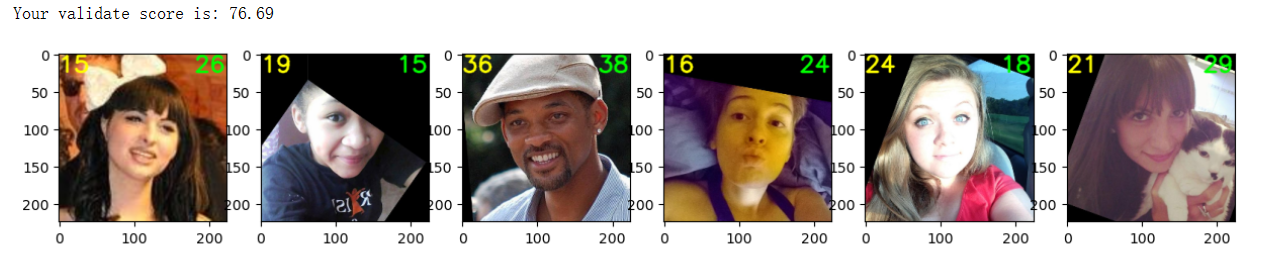
三、运行结果

（1）线性回归

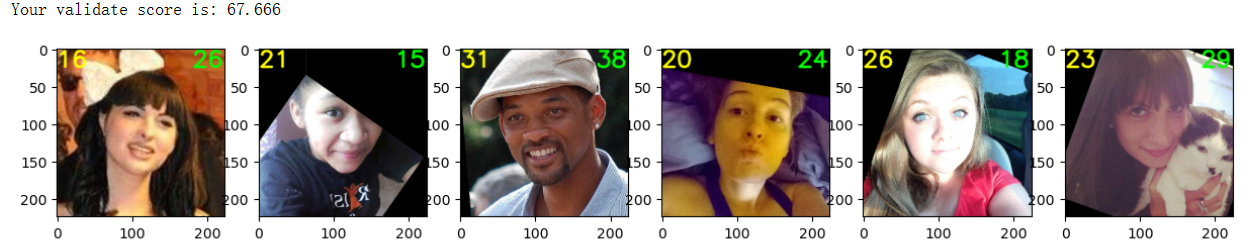
①解析解：



②梯度下降法：



③随机梯度下降法



（2）线性分类



四、补充代码

（1）线性回归

①解析解：

|  |
| --- |
| **def** closed\_form\_solution(age, features):  *# Preprocess* H = features  ones = np.ones(len(H))  H = np.column\_stack((ones, H))  Y = age   *#################################################################  #* ***TODO: YOUR CODE HERE*** *#################################################################   # calculate the closed form solution* weights = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(H), H)), np.transpose(H)), Y)   *# separate the weights and bias* bias = weights[0]  weights = weights[1:]    **return** weights, bias |

②梯度下降法：

|  |
| --- |
| **def** gradient\_descent(age, feature):  **assert** len(age) == len(feature)   *# Init weights and bias* weights = np.random.randn(2048, 1)  bias = np.random.randn(1, 1)    *# Learning rate* lr = 10e-3   **global** features\_val, age\_val  best\_loss = float(**'inf'**)  best\_weights = weights  best\_bias = bias  m = len(age)    **for** e **in** range(epoch):  *#################################################################  #* ***TODO: YOUR CODE HERE*** *#################################################################   # forward pass* age\_pred = np.dot(feature, weights) + bias   *# calculate loss* diff = age\_pred - age.reshape(-1, 1)    *# calculate gradient* grad\_w = np.dot(np.transpose(feature), diff) / m  grad\_b = np.sum(diff) / m   *# update weights* weights -= lr \* grad\_w  bias -= lr \* grad\_b    **if** momentum:  **pass**  *# You can also consider the gradient descent with momentum* loss\_val = np.sum(np.square(np.dot(features\_val, weights) + bias - age\_val.reshape(-1, 1))) / (2 \* len(age\_val))  **if** loss\_val < best\_loss:  best\_loss = loss\_val  best\_weights = weights  best\_bias = bias   **return** best\_weights, best\_bias |

③随机梯度下降法

|  |
| --- |
| **def** stochastic\_gradient\_descent(age, feature):  *# check the inputs* **assert** len(age) == len(feature)    *# Set the random seed* np.random.seed(0)   *# Init weights and bias* weights = np.random.rand(2048, 1)  bias = np.random.rand(1, 1)   *# Learning rate* lr = 10e-5   *# Batch size* batch\_size = 16    *# Number of mini-batches* t = len(age) // batch\_size   **global** features\_val, age\_val  best\_loss = float(**'inf'**)  best\_weights = weights  best\_bias = bias   **for** e **in** range(epoch\_sgd):  *# Shuffle training data* n = np.random.permutation(len(feature))    **for** m **in** range(t):  *# Providing mini batch with fixed batch size of 16* batch\_feature = feature[n[m \* batch\_size: (m+1) \* batch\_size]]  batch\_age = age[n[m \* batch\_size: (m+1) \* batch\_size]]    *#################################################################  #* ***TODO: YOUR CODE HERE*** *#################################################################   # forward pass* age\_pred = np.dot(batch\_feature, weights) + bias   *# calculate loss* diff = age\_pred - batch\_age.reshape(-1, 1)   *# calculate gradient* grad\_w = np.dot(np.transpose(batch\_feature), diff) / batch\_size  grad\_b = np.sum(diff) / batch\_size   *# update weights* weights -= lr \* grad\_w  bias -= lr \* grad\_b    **if** momentum:  **pass**  *# You can also consider the gradient descent with momentum* print(**'=> epoch:'**, e + 1, **' Loss:'**, round(loss, 4))   loss\_val = np.sum(np.square(np.dot(features\_val, weights) + bias - age\_val.reshape(-1, 1))) / (2 \* len(age\_val))  **if** loss\_val < best\_loss:  best\_loss = loss\_val  best\_weights = weights  best\_bias = bias   **return** best\_weights, best\_bias |

（2）线性分类

|  |
| --- |
| *# Define linear perceptron class* **class** PrimalPerceptron(object):  **def** \_\_init\_\_(self, x, y, w=**None**, b=**None**):  num\_sample, num\_dims = x.shape  np.random.seed(0)  *####################################  #* ***TODO: YOUR CODE HERE： init weights*** *####################################* **if not** w:  w = np.random.rand(1, num\_dims)  **if not** b:  b = np.random.rand(1)  self.x, self.y, self.w, self.b = x, y, w, b  self.lr = 0.1    **def** predict(self):  *####################################  #* ***TODO: YOUR CODE HERE, forward*** *####################################* preds = np.dot(self.x, np.transpose(self.w)) + self.b  y\_hat = np.sign(preds)  **return** preds, y\_hat   **def** update(self):  *####################################  #* ***TODO: YOUR CODE HERE, backward*** *####################################  # update the weights and bias* preds, y\_hat = self.predict()  **for** i **in** range(self.x.shape[0]):  **if** y\_hat[i, 0] \* self.y[i, 0] < 0:  self.w += self.lr \* self.y[i, 0] \* self.x[i, :]  self.b += self.lr \* self.y[i, 0]  **return** |