Ecperiment1-1: Linear Regression

Time: 2022/3/5
Location: Science_Building_119
Name: 易弘睿
Number: 20186103

Part I: Review

对上课时线性回归内容做一个简单的review,考虑只有一类特征的情况。首先进行正向传播，利用初始化后的权重系数w和偏置系数b计算y预测值,再计算Loss值。y预测值的计算公式如下: $\hat{y} = w^T x + b$， Loss值利用最小二乘法（MSE Loss function）计算公式如下: $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$， 其等价于 $\frac{1}{2} * ((w^T - x) - y)^2$。 计算w和b的梯度的公式如下： $dw = x(\hat{y} - y)$， $db = \hat{y} - y$。

Part II: Introduction

对初始代码的修改主要如下:
1. 对代码按照不同部分功能进行命名;
2. 对代码进行每行注释;
3. 增加sklearn自带的线性拟合工具进行w、b值的计算。

Part III: Annotation

1. 数据库的导入

In [1]:

```
import numpy as np                      # 加载numpy库进行基础数值计算
from sklearn.datasets import load_iris # 加载load_iris库调用鸢尾花卉数据集
import matplotlib.pyplot as plt         # 加载matplotlib.pyplot库完成作图
```

2. 数据集的制作

In [2]:

```
iris = load_iris()          # 调用鸢尾花卉数据集指向iris变量
data = iris['data']         # 调用数据集里的data
X = data[:,2].reshape(1,-1) # 将花瓣长度数据列向量取出并转置为一行
Y = data[:,3].reshape(1,-1) # 将花瓣长度数据行向量取出并转置为一行
```

3. 权重系数、偏置系数、学习率的初始化

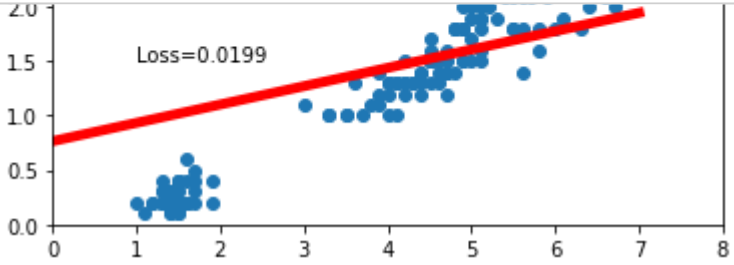
In [3]:

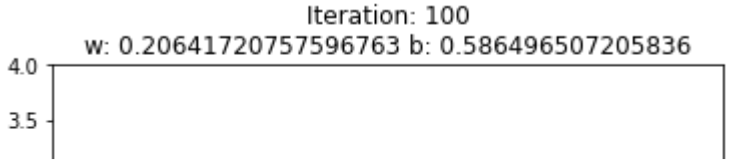
```
w = np.random.rand(1,1) # 初始化weight系数w的值，生成[0,1)，不包括1的1*1随机数矩阵
b = np.random.rand(1,1) # 初始化bia系数b的值，生成[0,1)，不包括1的1*1随机数矩阵
lr = 0.01               # 初始化学习率将初始值设置为0.01
# 原先没有设置最大循环次数，这里设置最大循环次数
max_iteration = int(1e3)
```

4. 梯度下降的执行

In [4]:

```
for iteration in range(10000):  # 设置10000次循环
    y_pred = np.dot(w.T, X) + b # y预测值的计算

    loss = (0.5 * np.dot((Y - y_pred).T,(Y - y_pred))).mean() # Loss值的计算
    w = w - lr * (X * (y_pred-Y)).mean()                      # 权重系数的更新
    b = b - lr * (y_pred-Y).mean()                            # 偏置系数的更新

    if iteration % 100 == 0:    # 每循环100次执行一次画图操作

        plt.scatter(X, Y, label="True Value") # 画散点图 scatter(x, y, 点的大小, 颜色, 标记)
        x = [i for i in range(8)]              # x范围大小0-8 也可以用list(range(8))
        y = [i*w[0][0] + b[0][0] for i in x]  # y值的计算
        plt.plot(x,y,'r-',lw=5)               # 利用迭代过程中的w、b的值plot每100次迭代时的拟合曲线
        plt.text(1, 1.5, 'Loss=%.4f' % loss)  # 显示每100次迭代的Loss值大小
        plt.xlim(0, 8)                        # 设定x座标轴的范围
        plt.ylim(0, 4)                        # 设定y座标轴的范围
        plt.title("Iteration: {}\nw: {} b: {}".format(iteration, w[0][0], b[0][0])) # 给图片添加标题
        plt.pause(0.5)                        # 每绘制一次暂停0.5秒

        # if MSE < 0.0000001, then stop training
        if loss < 0.0000001:
            break
```





5. 结果的打印

In [5]:

```
print("The value of w",w) # 打印拟合完成后的权重系数w
print("The value of b",b) # 打印拟合完成后的偏重系数b
```

```
The value of w [[0.415241]]
The value of b [[-0.36074208]]
```

Part IV: Use of sklearn

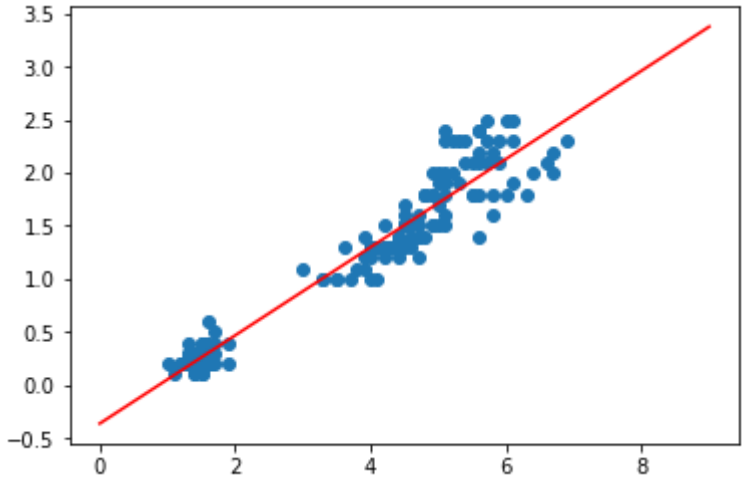
In [6]:

```
from sklearn.linear_model import LinearRegression
```

In [7]:

```
X = data[:,2].reshape(-1,1)          # 将花瓣长度数据列向量取出并转置为一行
Y = data[:,3].reshape(-1,1)          # 将花瓣宽度数据列向量取出并转置为一行
reg = LinearRegression().fit(X, Y) # 函数直接进行线性拟合
print(reg.coef_)                     # 打印斜率w的值
print(reg.intercept_)                # 打印截距b的值
x = np.linspace(0, 9, num=10)        # 在[0,9]区间内均匀产生10个数
y = reg.coef_*x+reg.intercept_       # 根据拟合曲线计算上一行10个数对应的函数值
y = y.reshape(10,-1)                 # 与转置等效，个人觉得此句可更改为直接转置，因为reshape需要输入行
plt.scatter(data[:,2],data[:,3])   # 画出数据集散点图
plt.plot(x,y,'r-')                   # 画出拟合曲线
plt.show()
print(x)
print(y)
```

```
[[0.41575542]]
[-0.36307552]
```




```
[0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
[[-0.36307552]
 [ 0.0526799 ]
 [ 0.46843531]
 [ 0.88419073]
 [ 1.29994614]
 [ 1.71570156]
 [ 2.13145698]
 [ 2.54721239]
 [ 2.96296781]
 [ 3.37872323]]
```

Part V: Conclusion

关于sklearn的LinearRegression

适用于非常大的数据集（训练其他模型不太可行），对稀疏数据也很有效，也适用于高维数据。
如果特征数量大于样本数量，线性模型的表现通常都很好。 LinearRegression对数据存在过拟合，Ridge是一种约束更强的模型，更不容易过拟合，Lasso使某些系数刚好为0，更容易解释。

1. 优点

线性模型的训练速度非常快，预测速度也很快。可以推广到非常大的数据集，对稀疏数据也有效。如果数据包含数十万甚至上百万个样本，可能需要研究如何使用LogisticRegression和Ridge模型的solver='sag'选项，在处理大型数据时，这一选项比默认值要更快。 用了我们之间见过的用于回归和分类的公式，理解如何进行预测是相对比较容易的。

2. 缺点

往往不完全清楚系数为什么是这样的。如果数据集中包含高度相关的特征，很难对系数做出解释。 在更低维的空间中，其他模型（SVM等）的泛化性能可能更好。