现在忘掉Sklearn吧，来自己手写一个人工智能线性回归，人工智能学习实验----01

原创

hhh江月 于 2021-08-06 16:12:10 发布

分类专栏： 人工智能 AI python 文章标签： 人工智能 机器学习 深度学习 python 可视化

本文链接：https://blog.csdn.net/m0_54218263/article/details/119426495

人工智能学习实验----01

本文是我的第一次人工智能学习的实验的操作，还请多多包涵。

现在忘掉Sklearn吧，来自己手写一个人工智能线性回归吧！！！—人工智能学习实验----01

模块化的编程固然很吃香，但是想要深入的发展，还是需要明白底层原理是怎么样的，这样才可以进一步是自己得到提升的啦。

文章目录

本次实验我们做一个线性回归

我们将只是用numpy以及pandas模块进行代码书写，不会去用Sklearn的模块的哦。

忘掉Sklearn，手写一个线性回归

我们知道python有内置的sklearn模块，里面自己封装好了sklearn，可以帮助不知道内涵的同学们方便的线性回归，但是想要理解人工智能的本质，还是需要通过实践来自己用程序写出来这种简单的算法的哦。

一、思路解说：

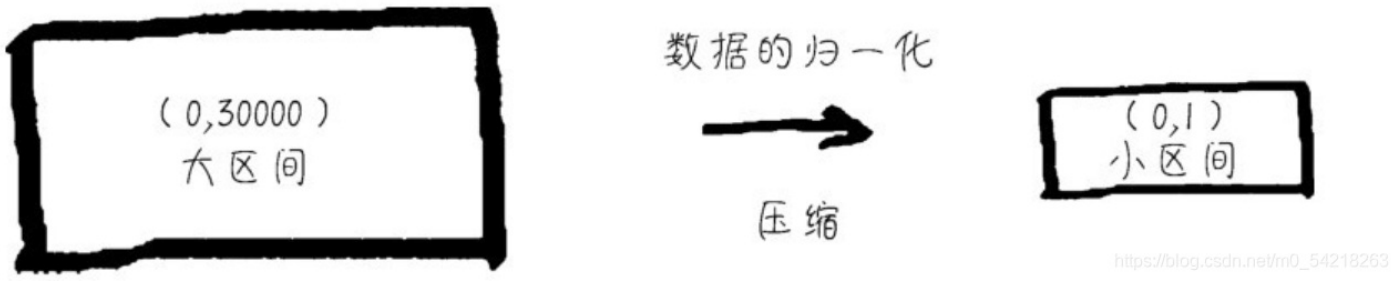
线性回归其实是非常常见也比较简单的一种回归的方式，而在机器学习中也是占据着重要的地位的，下面介绍具体的思路：

1、收集数据

为了进行回归，我们首先要去收集数据，一般需要大量的数据作为支撑，有时我们需要借助别人的接口来实现自己的功能，在此处，为简便期间，我们直接采用自己写一些数据来进行处理。

2、数据预处理

一般的数据预处理主要是使得我们的数据便于操作：
1）归一化：
这个方法可以使得很大的数据变得比较小，容易处理



2）其他操作：
例如转换数据类型等等，这是为了方便后面使用numpy或者pandas模块进行分析数据。

3、划分训练集以及测试集

一般而言，我们使用总的数据集的80%作为训练集，
另外的20%作为测试集。

4、进行线性回归

我们需要使用训练集数据进行模型的训练！！

在这里，我们不会去使用现有的线性回归模块，而是自己手写一个线性回归，我们将会采用梯度下降的方法来实现线性回归：
首先假设函数h（x）是结果，求出损失；

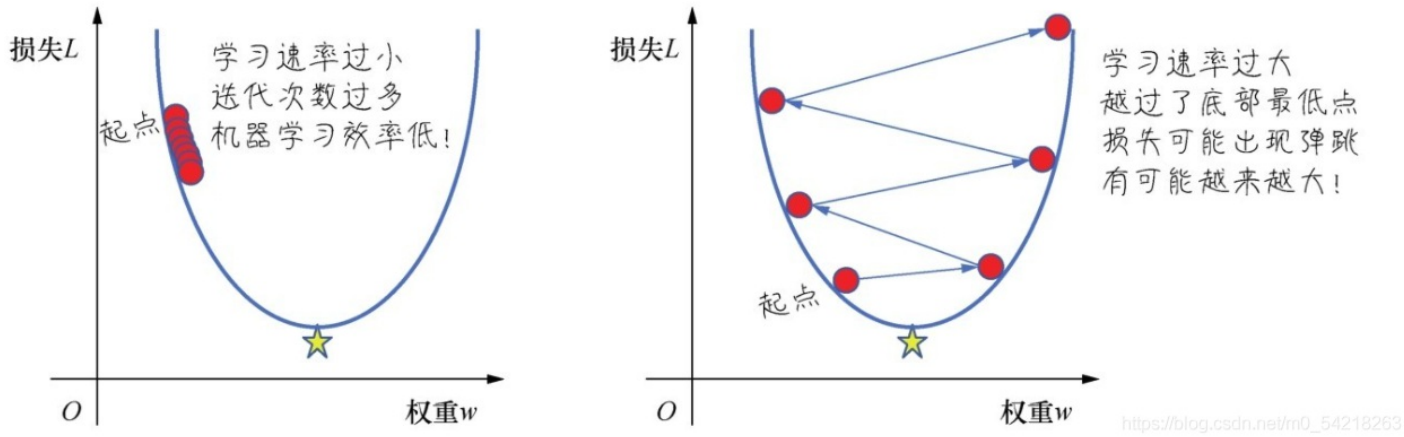
$$h\ (x)\ =wx+b$$

$$L(w,b)=MSE=\frac{1}{2N}\sum_{(x,y)\in D}(y-h(x))^2$$

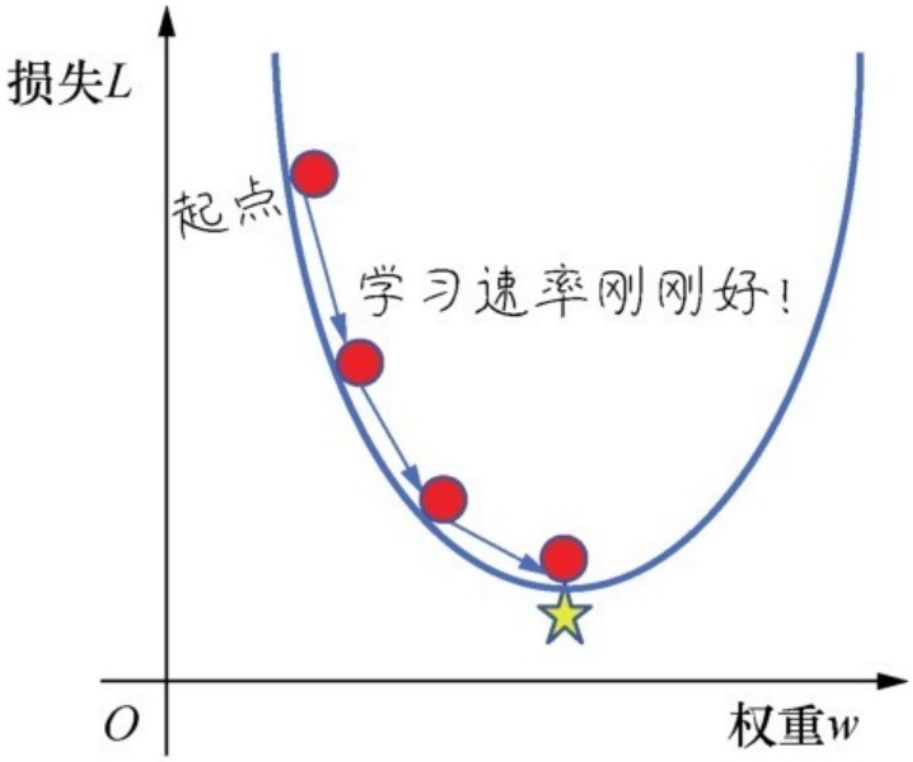
之后，我们对 w 以及 b 分别求导，根据导数的情况进行一定的调整：
也即是说，我们根据导数的正负来考虑 w 以及 b 向哪里移动可以达到理想的状态！！
并且以导数为依据来更新 w 以及 b 的数据；

$$w=w-\alpha\cdot\frac{\partial}{\partial w}L(w)$$

上式中，α 是叫做学习速率，这个也是一个机器学习中很关键的参数，当然，这个参数是由我们自己来设定的，但是它的取值，可能会影响到学习的次数多少以及效果的：
图片–1



图片–2



显然，我们希望学习速率是下面的 图片–2 所示的情况了啦。

因此，α（alpha） 这个参数是非常关键的啦。

每一次，对于 w 以及 b 进行更新以后，所得到的结果都应该是更加接近理想的数据的啦。

5、展示合成的曲线，并且进行评估打分

1）展示合成的曲线

我们可以使用可视化的模块展示原有的数据以及拟合之后的曲线来进行直观的比对

2）进行评估打分

我们可以使用测试集的数据对模型进行评估与打分

评估的话我们还是用图像直观的进行呈现，打分由于需要涉及更多的知识，我们谨在此列出来，不去做详细的打分操作了啦。

查准率（P值）是针对我们的预测结果而言的，它表示的是预测为正的样本中有多少是真正的正样本

查全率（R值）是针对我们原来的样本而言的，它表示的是样本中的正例有多少被预测正确了

查准率 P与查全率 R 分别定义为

$$P = \frac{TP}{TP + FP},$$

$$R = \frac{TP}{TP + FN}.$$

查准率和查全率是一对矛盾的度量.一般来说，查准率高时，查全率往往偏低;而查全率高时，查准率往往偏低。

F1-score，是统计学中用来衡量二分类模型精确度的一种指标。它同时兼顾了分类模型的准确率和召回率。F1分数可以看作是模型准确率和召回率的一种加权平均，它的最大值是1，最小值是0。

随着阈值的变化，就像假设检验的两类错误一样，如下图所示召回率和精确率不能同时提高，因此我们就需要一个指标来调和这两个指标,于是人们就常用F1-score来进行表示：

$$F1 = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{\text{样例总数} + TP - TN}.$$

二、代码实现：

下面的代码有详细的注释，这个就不需要多做说明了。大家看着注释也是一定能够明白的啦。

只有求导哪里，我们将求导转换为了矩阵乘积这个需要注意一下：

1、

用数学语言描述梯度计算过程如下：

$$梯度=\frac{\partial}{\partial w}L(w)=\frac{\partial}{\partial w}\frac{1}{2N}\sum_{(x,y)\in D}(y-h(x))^2=\frac{1}{2N}\sum_{(x,y)\in D}(y-(w\cdot x))\cdot x$$

也可以写成

$$梯度=\frac{1}{2N}\sum_{i=1}^{N}(y^{(i)}-(w\cdot x^{(i)}))\cdot x^{(i)}$$

2、

引入学习速率之后，用数学语言描述参数w随梯度更新的公式如下：

$$w=w-\alpha\cdot\frac{\partial}{\partial w}L(w)$$

即

$$w=w-\frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)}-(w\cdot x^{(i)}))\cdot x^{(i)}$$

其他部分的理解应该没有太大的问题了啦。

```
import numpy as np
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt
import sklearn
import seaborn as sb
# visible


"""
here we will not use the module like sklearn above,
instead, we will write the liner regression by ourselves,
and we will use math to support our project!
"""

"""
1.get data
here we just use the data we write by ourselves
"""

X = pd.DataFrame([[-20.4], [-19.1], [-18.2], [-16.9], [-16.2], [-15.3], [-14.1], [-12.8],
```

```
X = pd.DataFrame([[-20.4], [-19.1], [-18.2], [-16.9], [-16.2], [-15.5], [-14.1], [-12.8],
                  [-12.1], [-11.2], [-9.6], [-9.3], [-8.1], [-7.4], [-6.1], [-4.9],
                  [-4.4], [-3.0], [-2.2], [-1.4], [-0.1], [1.1], [1.7], [3.2],
                  [4.4], [5.1], [5.8], [7.0], [8.1], [9.0]])
# print(X)
# X collection
# print()

y = pd.DataFrame([[-40], [-38], [-36], [-34], [-32], [-30], [-28], [-26], [-24], [-22],
                  [-20], [-18], [-16], [-14], [-12], [-10], [-8], [-6], [-4], [-2], [0],
                  [2], [4], [6], [8], [10], [12], [14], [16], [18]])
# print(y)
# y

"""
2.normalization the data
"""

max_of_X = X.max(axis=0)  # X's max
min_of_X = X.min(axis=0)  # X's min
X = (X - min_of_X) / (max_of_X - min_of_X)
# normalization the X
# print(X)

max_of_y = y.max(axis=0)  # y's max
min_of_y = y.min(axis=0)  # y's min
y = (y - min_of_y) / (max_of_y - min_of_y)
# normalization the y
# print(y)

"""
3.split the train and test
"""

train_X = X.loc[: len(X) * 0.8 - 1]
# print(train_X)
# 80%
train_y = y.loc[: len(y) * 0.8 - 1]
# print(train_y)
# 80%
test_X = X.loc[len(X) * 0.8:]
# print(test_X)
# 20%
test_y = y.loc[len(y) * 0.8:]
# print(test_y)
# 20%

"""
4.use the gradient descent to finish the liner regression
gradient descent
"""


def gradient_descent_to_get_w_b(tra_x, tra_y, w, b, alpha):
    """
    use the gradient descent method to get the proper w and b
    :param tra_x: train_x
    :param tra_y: train_y
    :param w: w
    :param b: b
```

```
    :param alpha: alpha
    :return: w, b
    """
    # guess the w and the b
    h = tra_x * w + b
    # Attention!
    # the calculation is the vector calculation
    loss = h - tra_y
    # print(loss)
    # print(len(loss))
    # attention that there is tra_y but not y !!
    # calculate the loss of every data (vector calculation)

    # derivative the w: T and dot
    # derivative_w = tra_x.T.dot(loss) / len(tra_x)
    derivative_of_w = tra_x.T.dot(loss) / len(tra_x)
    # derivative the b: sum


    # sum finally is right!!
    sum_of_loss = 0
    for v in range(len(loss)):
        sum_of_loss += loss.loc[v]
    derivative_of_b = sum_of_loss / len(tra_y)

    # print(derivative_of_b)

    # print(sum(loss))

    # sum_of_loss = 0
    # calculate by myself

    # for v in range(len(loss)):
    #     sum_of_loss += loss.loc[v]
    # print("loss of sum is :", sum_of_loss)

    # update the w and the b:

    w = w - alpha * derivative_of_w
    # print(w)
    # w

    b = b - alpha * derivative_of_b
    # print(b)
    # b

    # print(w)
    # print()
    # print(b)
    # print()
    # print(w.loc[0, 0])
    # print()
    # print(b.loc[0])

    return w.loc[0, 0], b.loc[0]


"""
5.run the code to see what will happen
at the same time
```

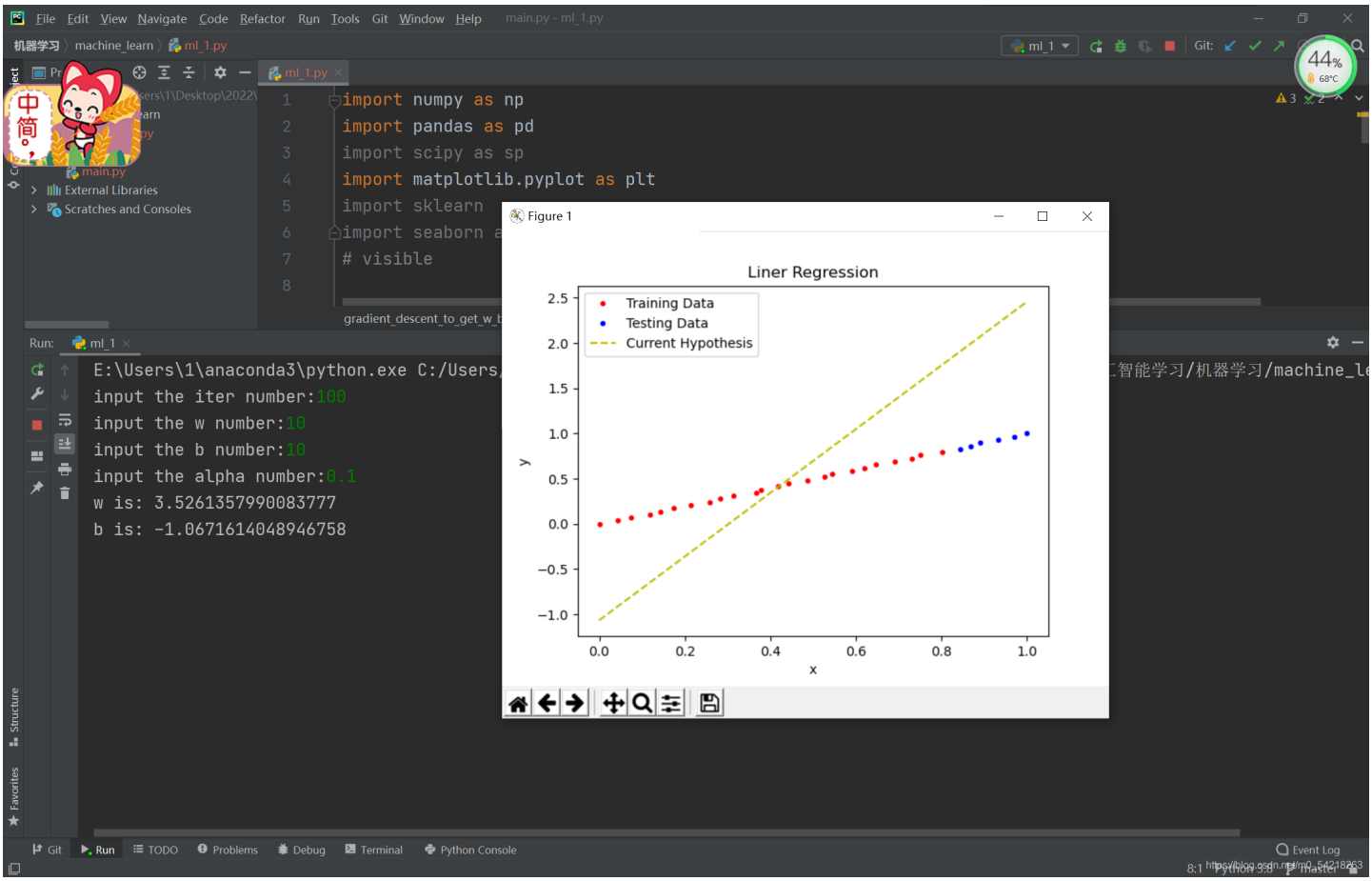
```
at the same time
we will use the matplotlib to show the data
"""

if __name__ == '__main__':
    """
    main
    """

    # input the params
    iter_number_of_test = int(input("input the iter number:"))
    input_the_w = float(input("input the w number:"))
    input_the_b = float(input("input the b number:"))
    input_the_alpha = float(input("input the alpha number:"))

    for t in range(iter_number_of_test):
        input_the_w, input_the_b = gradient_descent_to_get_w_b(tra_x=train_X, tra_y=train_y,
                                                               w=input_the_w, b=input_the_b,
                                                               alpha=input_the_alpha)
        # circulating

    print("w is:", input_the_w)
    print("b is:", input_the_b)
    # show the w and the b data

    plt.title("Liner Regression")

    liner_x = np.linspace(X.min(), X.max(), 500)
    # spice
    liner_y = [input_the_w * x_of_liner + input_the_b for x_of_liner in list(liner_x)]
    # calculate the liner y data
    plt.plot(train_X.loc[0: len(train_X), 0], train_y.loc[0: len(train_y), 0], 'r.', label="Training Data")
    # show the train collection
    # print(test_X)
    # print(test_y)
    plt.plot(test_X.loc[len(train_X):, 0], test_y.loc[len(train_y):, 0], 'b.', label="Testing Data")
    # show the testing data

    plt.plot(list(liner_x), list(liner_y), 'y--', label="Current Hypothesis")
    # show the line



    plt.xlabel("x")
    plt.ylabel("y")
    # plt.xlabel("x")
    # plt.ylabel("y")
    plt.legend()
    # use the legend
    plt.show()
    # show all the things
```

运行上述代码即可

三、运行结果

我们进行两次测试：

1、

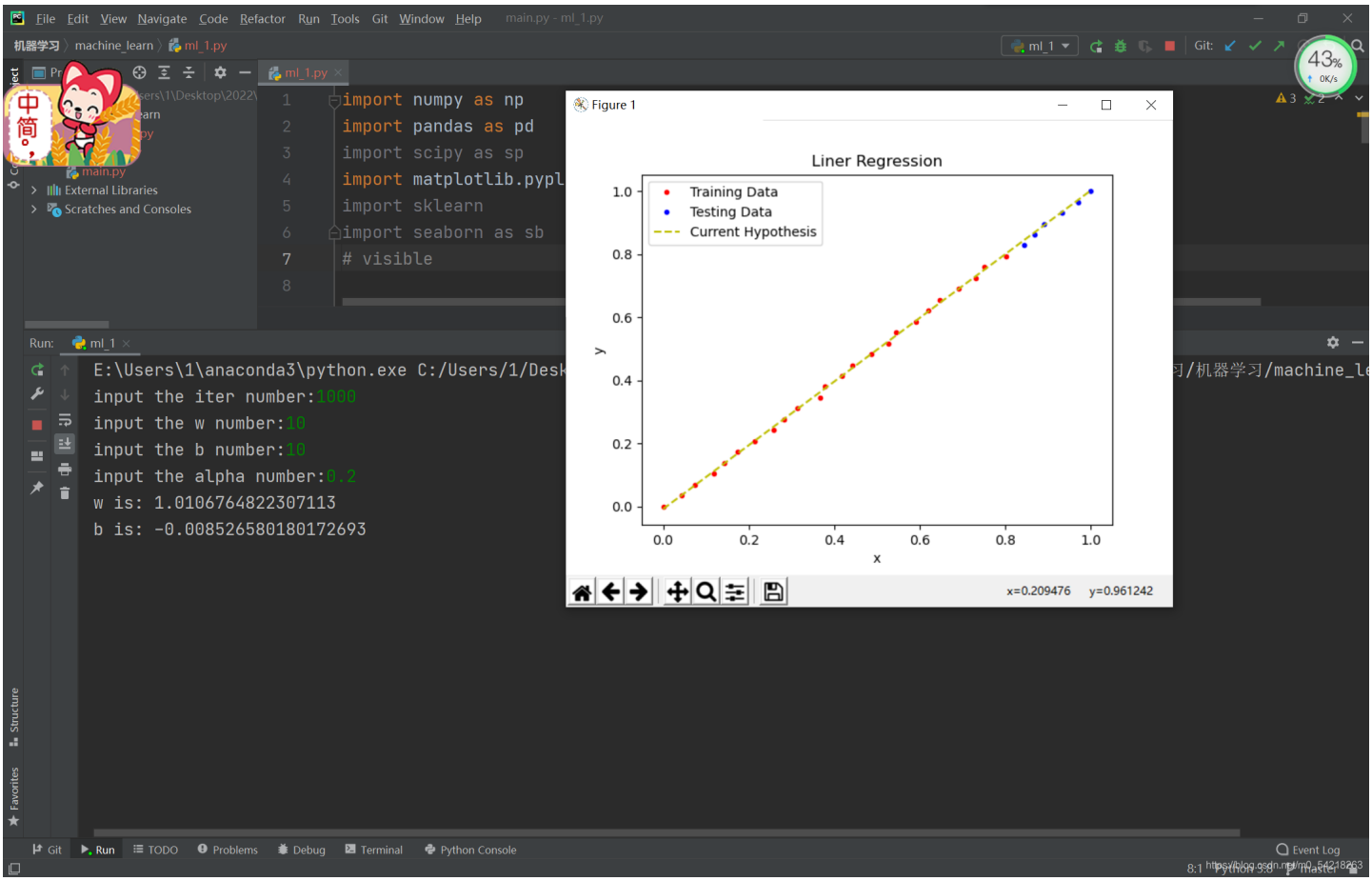


这里，我们设置了一些参数，我们发现这里不太吻合，原因在于，学习速率以及学习的次数选择不合适

（红色是训练集，蓝色是测试集，黄线是拟合曲线）

下面，我呢更改学习速率，以及学习次数，如下图；

2、



这一次吻合的就比较好了

从而说明这两个参数的选取还是很重要的，后面的实验我们会进一步探究，学习速率以及学习次数对于机器学习的影响。

本次的实验到此结束，希望对大家有一点的启发了啦。

谢谢大家。

同时也希望大家持续关注后续的实验了啦。