机器学习作业2-线性回归

一元线性回归

根据dataset_regression.csv文件求得最小二乘解，在这个数据集中只有一个特征向量的输入x，使用最小二乘法做一元回归让其拟合y即可。

线性预测的方程一般形式是：

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \ldots \theta_n x_n = \theta^T X$$

目标是损失函数最小，损失函数的形式是

$$J_N(\theta) = \sum Loss(y^{(i)}, f(x^i;\theta))$$

通过求偏导数得到最小二乘解的形式是

$$\frac{1}{N}\sum \theta_0 + \theta_1 x^i - y^i = 0; \frac{1}{N}\sum(\theta_0 + \theta_1 x^i - y^i)x^i = 0$$

其中θ1是

$$\theta_1 = \frac{\sum x^i y^i - N\overline{xy}}{\sum (x^i)^2 - N\overline{x}^2}$$

加载数据：

```
import pandas as pd
X=pd.read_csv("H:\ml-hw2\dataset_regression.csv" )
```

按照上面的方程求解

```
def linear_regression(x,y):
    N = len(x)
    sumx = sum(x)
    sumy = sum(y)
    sumx2 = sum(x**2)
    sumxy = sum(x*y)
    A = np.mat([[N,sumx],[sumx,sumx2]])
    b = np.array([sumy,sumxy])
    return np.linalg.solve(A,b)
```

数据集中总共9个数据，选取前面7个作为训练集，将他们放入上面的函数进行拟合，得出一元线性回归的结果。

```
a0,a1 = linear_regression(x,y)
```

```
a0
```

```
0.2964285714285717
```

```
a1
```

```
2.278571428571429
```

计算训练误差

```
for i in range(7):
    print((-2+0.5*i)*a1+a0-y_t[i])
```

```
0.6392857142857142
-0.6214285714285719
-1.4821428571428572
0.6571428571428573
1.2964285714285717
0.6357142857142861
-1.1249999999999996
```

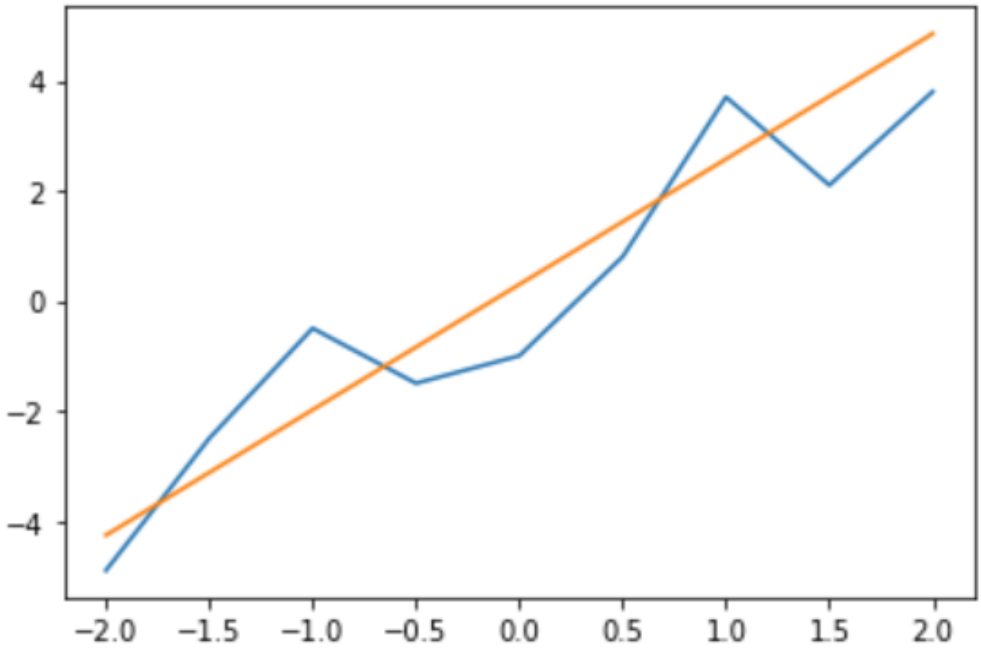
将预测数据和最后两个数字进行对比，计算预测误差。

```
print(1.5*a1+a0-y_t[7])
print(2*a1+a0-y_t[8])
```

```
1.6142857142857152
1.0535714285714297
```

打印预测结果：

```
import matplotlib.pyplot as plt
plt.plot(x_t,y_t)
plt.plot(x_l,y_l)
plt.show()
```



多元线性回归

导入第二部分的数据winequality-white.csv，因为这次的数据有11个特征，因此需要用到多元线性回归。多元线性回归使用了梯度下降法，对于θ进行调优，调优过程是：

$$\theta_j = \theta_j + \alpha\frac{1}{m}\sum(y^i - h_\theta(x^i))x_j^i$$

重复这个过程直到收敛，将其化解伟向量型表示，可以利用到并行计算优化性能。

将上面的式子用代码表示，用下面这个函数计算梯度下降

```
def bgd(alpha, maxloop, epsilon, X, Y,xt,yt):

    m,n = X.shape
    theta = np.zeros((n,1))

    count = 0 # 记录迭代轮次
    converged = False # 是否已经收敛的标志
    error = np.inf
    errors = [Loss(theta, X, Y),] # 记录每一次迭代得代价函数值
    error_t=[]
    print(errors)
    while count<=maxloop:
        if(converged):
            break
        count+=1
        for j in range(n):
            deriv = sum(np.dot(X[:,j].T, (h(theta, X) - Y)))/m
            theta[j] = theta[j] - alpha*deriv
        error = Loss(theta, X, Y)
        pred=np.dot(xt,theta)
```

```
        error_t.append(np.dot((pred-yt).T,(pred-yt))[0])
        print(error)
        errors.append(error)
        if(abs(errors[-1] - errors[-2]) < epsilon):
            converged = True
    return theta,errors,error_t
```

在实际的训练中，我们加载数据之后需要将其划分为训练集和测试集：

```
data=pd.read_csv("H:\\ml-hw2\\winequality-white.csv")
X=data.drop(['quality'],axis=1)
Y=data.iloc[:,[11]]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=5)
```

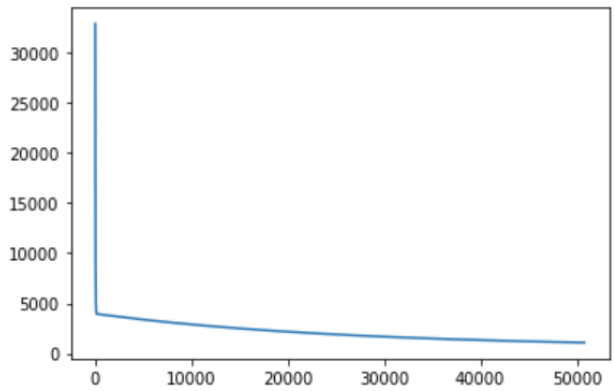
训练的时候发现学习率α需要取一个很小的数值

t,e,et=bgd(0.000001,100000,0.00001,x_tn,y_tn,x_ts,y_ts)。

在几万次迭代收敛之后，测试集上的均方误差mse大概在一千左右。

下面是均方误差迭代下降的曲线。

```
In  [60]: plt.plot(et)
Out[60]: [<matplotlib.lines.Line2D at 0x2e7390e8208>]
```

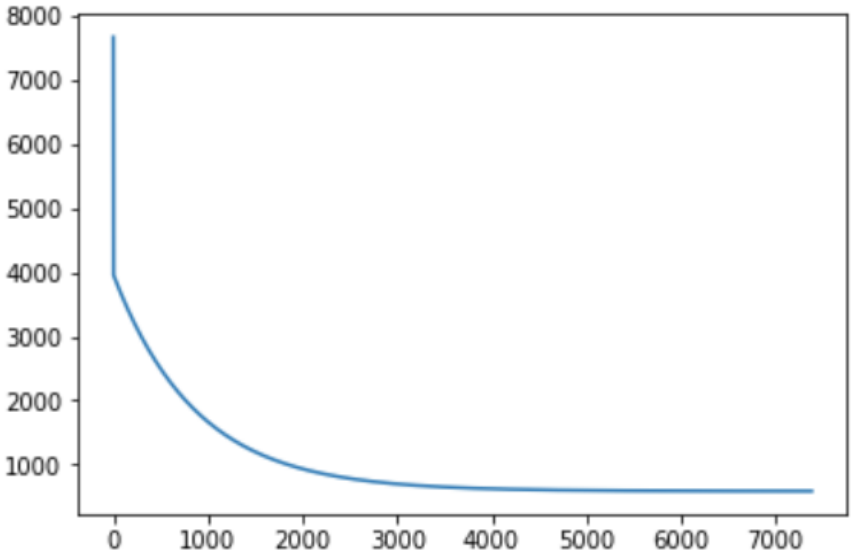


通常来说，学习率越低越好，但是更低的学习率也意味着梯度下降的速度会更慢，需要迭代更多的次数，如果限制迭代次数的话，那么应该可以找到一个最合适的学习率。

经过几次测试，0.00003的学习率是相对合适的，在相同的迭代条件下，能够比更低的学习率得到更好的均方误差。

In [79]:
```
plt.plot(et)
```

Out[79]:
```
[<matplotlib.lines.Line2D at 0x2e7391c5548>]
```



岭回归实现

岭回归通过给模型参数添加高斯先验概率来鼓励其取绝对值较小的数值，称为L2正则化或权重衰减。岭回归的损失函数相比线性回归多了正则化项。

$$J(\theta) = \frac{1}{2m}\sum(h_\theta(x^i) - y^i)^2 + \lambda\sum\theta_j^2$$

编程实现：

```
x_T = np.transpose(x_tn)
num=20000
k_mat = np.linspace(-10000, num-1-10000, num=num)
beta = np.zeros([num, 11])
xMat = np.mat(x_tn)
yMat = np.mat(y_tn).T
iner = np.zeros((np.shape(xMat)[1],np.shape(xMat)[1]))
weights = np.zeros((np.shape(xMat)[1],np.shape(yMat)[0]))
for k in range(num):
    xTx = xMat.T * xMat
    iner = xTx +  np.eye(np.shape(xMat)[1]) * k_mat[k]
    weights = iner.I * ( yMat*xMat).T
    beta[k,:] = weights.T
```

经过两万次迭代训练之后，在3918行训练集上的训练误差约为2768，在980行测试集上的测试误差约为624.6。

```
pred_tn=np.dot(x_tn,beta[19999])
```

```
np.dot(pred_tn-y_tn.T,(pred_tn-y_tn.T).T)
```

```
array([[2768.60150987]])
```

```
pred_ts=np.dot(x_ts,beta[19999])
np.dot(pred_ts-y_ts.T,(pred_ts-y_ts.T).T)
```

```
array([[624.61178477]])
```