使用线性回归预测鲍鱼的年龄

首先说一下评分函数我是怎么计算出来的

因为这个不是分类，不是像上次一样的分成`Iris-setosa`等类，也不是输出`0`或者`1`，所以不能用分类精度来作为评分的依据，毕竟真实年龄是 9 岁，我算出来 8.98 岁也是完全可以接受的

助教说，用均方误差来判断

$$MSE = \frac{SSE}{n} = \frac{1}{n} \times \sum_{i=1}^{n} \times w_i (y_i - \hat{y}_i)$$

均方误差是越小越好，最低是 0 ，大却可以大到无穷大，所以不能作为Checker的评分函数

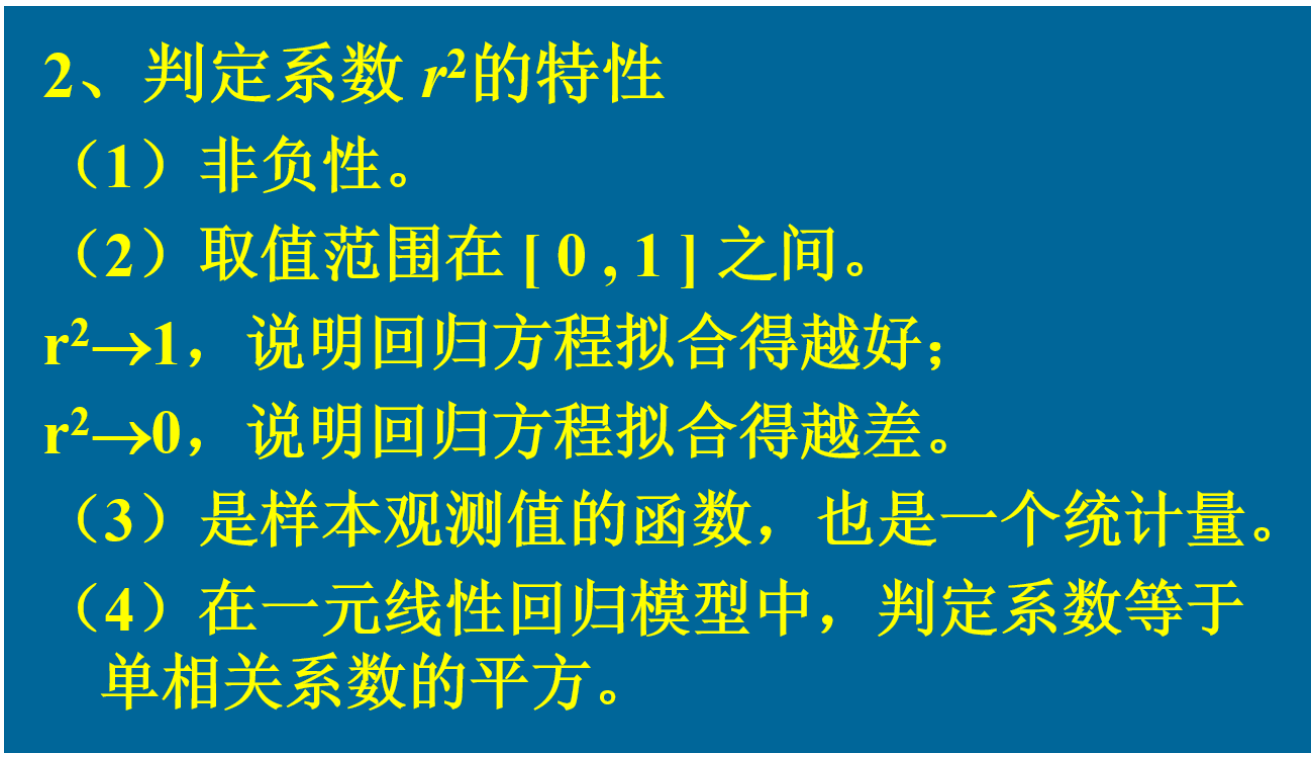
期望的得分是在 $[0, 100]$ 之间，所以均方误差在 $[0, \infty]$ 是不行的，同时，得分希望是越高越好，所以均方误差越小越好是不行的

那么这里就需要用到概率论的知识

判定系数是回归平方和占总离差平方和的比例。

$$r^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

反映回归直线的拟合程度。各样本观测点与样本回归直线靠得越紧，SSR在SST中所占的比例就越大，模型拟合程度越高。反之，拟合程度越差。



所以我想到的评分函数是

$$(1-\frac{\sum_i(\hat{y^{(i)}}-y^{(i)})^2}{\sum_i(\overline{y}-y^{(i)})^2})\times 100$$

然后在Checker中给出

> 如果你的判定系数在区间 $(0,1]$ 内，将会被等比例线性映射到区间 $(0,100]$ 并返回评分结果，结果精确到个位数；如果你的判定系数为负数、 0 或大于 1 ，则说明你的线性回归出现了错误，判题将返回 0 分。

这样一来，就满足了上面的条件

我是用最小二乘法来实现的

公式的推导从略，最后就是实现矩阵的运算

`numpy` 封装了矩阵这一类型，这就使得程序变得非常简单

而且，矩阵运算已经重载了运算符 `*` ，所以两个矩阵 `MatA` 、 `MatB` 可以直接用 `MatA * MatB` 来完成矩阵的乘法

矩阵求转置、求逆也是有对应方法的，所以，核心代码其实就是只有一句话

```
W = (X.T * X).I * X.T * Y
```

下面简单说说整个程序

```
while True:
    s = in_train.readline()
    if not s:
        break
    data = s.split(',') # 读入一行，以逗号分隔出各个字段
    x_i = []
```

```
    # 对性别的字符做处理，变成数值
    if data[0]=='F':
        data[0] = 0
    elif data[0] == 'M':
        data[0] = 1
    else:
        data[0] = -1
    # 依次添加到列表
    for i in range(x_len):
        x_i.append(float(data[i]))
    x.append(x_i) # X
    y.append(float(data[x_len])) # Y

X = np.mat(x) # 变成矩阵
Y = np.mat(y).T # 变成矩阵，并转置

W = (X.T * X).I * X.T * Y # 计算参数theta

# print(W)

while True:
    s = in_test.readline()
    if not s:
        break
    data = s.split(',')
    x_i = []
    if data[0]=='F':
        data[0] = 0
    elif data[0] == 'M':
        data[0] = 1
    else:
        data[0] = -1
    # 对于一组X
    for i in range(x_len):
        x_i.append(float(data[i]))
    X = np.mat(x_i)
    predict = X * W # 矩阵乘法，直接得到结果
    print(predict[0,0], file=out_predict)
```

但是准确率不高

试过了梯度下降算法，效果不好

这里用的优化方法是局部加权

也就是本来是要令$\sum_i (Y^{(i)} - \theta^T x^{(i)})^2$最小，现在是要令$\sum_i w^{(i)} (Y^{(i)} - \theta^T x^{(i)})^2$最小

$w^{(i)}$为权值，如果$w^{(i)}$很大，我们将很难去使得$\left(y^{(i)} - \theta^T x^{(i)}\right)^2$小，所以如果$w^{(i)}$很小，则它所产生的影响也就很小

通常

$$w^{(i)}$$

的取值为

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

那我也就用这个函数了

最后的θ为

$$\theta = (X^T W X)^{-1} X^T W y$$

然后用需要预测的X乘上θ就得到了$\hat{Y}$

下面对程序做修改如下

```python
import numpy as np
import math

f_train = open('train.txt', 'r')
f_test = open('test.txt', 'r')
f_predict = open('predict.txt', 'w')

# 处理输入，用逗号分隔，并将字符映射为数值，以列表的形式返回
def get_fv(s):
    fv = s.strip().split(',')
    if fv[0] == 'F':
        fv[0] = 0.1
    elif fv[0] == 'M':
        fv[0] = 0.2
    else:
        fv[0] = 0.3
    for i in range(len(fv)):
        fv[i] = float(fv[i])
    return fv

# 计算损失函数，这是在梯度下降法用到的
def get_loss(loss):
    loss **= 2
    return loss

# X和Y
x = []
y = []

# m记录一共有多少样本
m = 0

while True:
    s = f_train.readline()
    if not s:
        break
    m += 1
    # print(m)
```

```
    data = get_fv(s)
    x_i = [1.0] # 常数项β0的值
    n = len(data) # 字段个数
    for i in range(n - 1):
        x_i.append(data[i])
    # 添加进列表，然后加入矩阵
    x.append(x_i)
    # print("x_i",x_i)
    y.append(data[n - 1])

# X和Y变成矩阵形式，Y需要转置
X = np.mat(x)
# print("X", X)
Y = np.mat(y).T
# print("Y", Y)
# 得到初始的W
W = (X.T * X).I * X.T * Y

# print("W", W)

cnt = 1
# f_test = open('train.txt', 'r')
while True:
    s = f_test.readline()
    if not s:
        break
    print("now predicting", cnt)
    cnt += 1
    data = get_fv(s)
    x_i = [1]
    n = len(data)
    for i in range(n):
        x_i.append(data[i])
    ## XX是当前需要预测的X值
    XX = np.mat(x_i)
    weights = np.mat(np.eye(m))  # eye生成对角矩阵
    for i in range(m):
        diffMat = XX - X[i, :]  # 计算测试点坐标和所有数据坐标的差值
        # 计算权值 w = exp((- (xi-X) ^2)/(2*k*k))
        K = 0.6 #经过测试，K值取0.6~0.8效果可观
        weights[i, i] = np.exp(diffMat * diffMat.T / (-2.0 * (K ** 2)))  # 计算权重对角矩阵
    # 对x值进行加权计算
    theta = (X.T * weights * X).I * X.T * weights * Y # 用加权的公式计算θ参数
    predict = float((XX * theta)[0, 0]) # 用估计得到的θ来计算Y_hat
    print(predict) # 输出结果
    print(predict, file=f_predict) # 输出到文件
```

如上所示，局部加权以后，代码基本不用怎么变，之前的字符串分隔，然后处理、变成矩阵，都是一样的

最后就是计算权重的时候考虑一个加权

核心代码其实就只有几行

```
for i in range(m):
    diffMat = XX - X[i, :]  # 计算测试点坐标和所有数据坐标的差值
    # 计算权值 w = exp((- (xi-X) ^2)/(2*k*k))
    K = 0.6 #经过测试，K值取0.6~0.8效果可观
    weights[i, i] = np.exp(diffMat * diffMat.T / (-2.0 * (K ** 2)))  # 计算权重对角矩阵
    # 对x值进行加权计算
theta = (X.T * weights * X).I * X.T * weights * Y # 用加权的公式计算θ参数
predict = float((XX * theta)[0, 0]) # 用估计得到的θ来计算Y_hat
```

注释已经写清楚了，就是计算测试点与所有点的坐标的差值，也就是其实是考虑离自己近的点对自己的影响更大些，然后取一个K值，计算权重，最后就是带着权重去做估计

于是，问题转化为最无聊的**调参数**

我是没有怎么调，会了就可以了

只是从裸的算法的 57 分，优化到了 59 分，最后的测试发现，K 值在 $0.6 \rightarrow 0.8$ 之间的时候，效果还是可以的，更大和更小的值会使得判定系数变得特别小

我只是做了没几次

2018-10-15 17:39:22 Partial score: 59 → 1331695, Text
2018-10-15 17:24:08 Partial score: 59 → 1331675, Text
2018-10-15 17:21:01 Partial score: 58 → 1331671, Text
2018-10-15 17:11:23 Partial score: 58 → 1331662, Text
2018-10-15 11:37:20 Partial score: 57 → 1331399, Text

有同学持之以恒是可以优化到更高的分数的，例如这位同学就通过调整参数到了 60 分

Home | Problem | Contest | Current: jxtxzzw

Problem #279

Basics **Submits** Preview Revision 11 Revision 10 Revision 9 Revision 8 Revision 7

#	When	[redacted]	Lang	Verdict	Time	Rejudge	IP
1331181*	2018-10-14 20:35:32	[redacted]	Text	Partial score: 56	0.004	Rejudge	180.160.59.107
1331175*	2018-10-14 20:27:36	[redacted]	Text	Partial score: 60	0.000	Rejudge	180.160.57.180
1331163*	2018-10-14 20:23:17	[redacted]	Text	Partial score: 60	0.004	Rejudge	180.160.57.180
1331157*	2018-10-14 20:20:02	[redacted]	Text	Partial score: 60	0.000	Rejudge	180.160.60.207
1331155*	2018-10-14 20:16:13	[redacted]	Text	Partial score: 59	0.000	Rejudge	180.160.58.209
1331153*	2018-10-14 20:10:28	[redacted]	Text	Partial score: 58	0.000	Rejudge	180.160.57.180
1331145*	2018-10-14 20:03:07	[redacted]	Text	Partial score: 0	0.000	Rejudge	180.160.58.248
1328678*	2018-10-11 22:28:35	[redacted]	Text	Partial score: 0	0.000	Rejudge	172.20.145.82
1328547*	2018-10-11 20:45:26	[redacted]	Text	Partial score: 56	0.000	Rejudge	172.20.145.82
1328535*	2018-10-11 20:41:36	[redacted]	Text	Partial score: 57	0.000	Rejudge	172.20.145.82
1328288*	2018-10-11 17:52:53	[redacted]	Text	Partial score: 57	0.000	Rejudge	180.160.57.180
1328257*	2018-10-11 17:26:09	[redacted]	Text	Partial score: 56	0.000	Rejudge	180.160.48.87

« ‹ 1 › »

但是，受限于算法，这本身已经不太可能有更大提高了

毕竟参数调整只是小的优化，即便再交上几十次，提高起来也不过是只有几分，所以我不想花时间在这个上面了

另外需要一提的是，我在网上看到有这样一个公式

这个公式的真实性没有验证过，也没有去仔细看这个公式的推导

$$\log(rings) = \beta_0 + \beta_1(\mathrm{Sex}_i) + \beta_2 \log(\sqrt[3]{\mathrm{l}*\mathrm{w}*\mathrm{h}_i}) + \beta_3(\mathrm{Shell}_i) + \beta_4(\mathrm{Shucked}_i)$$

如果这个公式真的属实的话，那么这就不是简单的线性回归能够预测的准了，所以得到 0.6 的准确率我已经很满足了

附 OJ 结果

知识分析与应用基础（2018年秋）

Dashboard My Status **Standings** Balloon Manage

Refresh

#	Who	=	A 100	B 100	C 100
	* Rooobin	259	100	96	63
1	jxtxzzw 10165102154 张臻炜	256	100	97	59
1	10165102128 10165102128 张雨时	256	100	97	59
1	10152150124 10152150124 江心雨	256	100	97	59
1	nice666 10165102232 江南	256	100	97	59
5	10165102139 10165102139 刘金昊	255	100	95	60
6	10165102101 10165102101 邓贵强	253	100	97	56
7	flyingman11 10165102151 方炜	251	100	92	59
8	10165102204 10165102204 高桢	200	100	100	

2018年10月20日

迷之优化：

K调成0.25

同时判断，对超过平均值的预测值，四舍五入精确到整数，对小于平均值的预测值，保留原值

神仙操作，得分有较大提高，从原来的592分，到了609分

2018-10-20 23:34:10 Partial score: 609 → 1336722, Text
2018-10-20 23:04:02 Partial score: 595 → 1336706, Text
2018-10-15 17:39:22 Partial score: 592 → 1331695, Text
2018-10-15 17:24:08 Partial score: 586 → 1331675, Text
2018-10-15 17:21:01 Partial score: 577 → 1331671, Text
2018-10-15 17:11:23 Partial score: 581 → 1331662, Text
2018-10-15 11:37:20 Partial score: 567 → 1331399, Text

#	Who	=	A 1000	B 1000	C 1000
	* Rooobin	2601	**1000**	**968**	**633**
1	jxtxzzw 10165102154 张臻炜	2575	**1000**	**966**	**609**
2	ohyummy 10165102127 陈芷萱	2572	**1000**	**978**	**594**
3	10165102128 10165102128 张雨时	2570	**1000**	**968**	**602**
4	10165102204 10165102204 高桢	2563	**1000**	**1000**	**563**
5	10152150124 10152150124 江心雨	2560	**1000**	**966**	**594**
6	10165102101 10165102101 邓贵强	2559	**1000**	**968**	**591**
7	nice666 10165102232 江南	2558	**1000**	**966**	**592**
8	10165102139 10165102139 刘金昊	2547	**1000**	**948**	**599**
9	ultmaster 10165102136 张羽戈	2526	**1000**	**962**	**564**
10	hellwrld17 10165102250 裴胜军	2523	**1000**	**936**	**587**
11	flyingman11 10165102151 方炜	2518	**1000**	**924**	**594**