MS328 Assignment1

周李韬 5180309104O7

2020 年 3 月 22 日

线性模型的实际数据分析: 对于 Galton 数据集，基于第二章介绍的各种重抽样方法比较两种回归模型的效果

1 数据处理与模型设置

首先导入数据集, 数据集格式如下所示. 我们用到的数据是每一行的第二、三列的父母身高，第七列的孩子性别，第八列的孩子身高。身高单位为 cm。女性身高已经乘以 1.08 的系数以消除性别的影响。

[1]:
```
import numpy as np
import pandas as pd
data = pd.read_csv('GaltonHeight.csv',header=0,index_col=0)
data.values[0]
```

[1]:
```
array(['001', 199.39, 183.7944, 75.43, 4, 1, 'male', 185.928],
      dtype=object)
```

[2]:
```
def lg(x,y):
    b = np.ones(x.shape[0])
    x1 = np.c_[b,x]
    return np.matmul(np.matmul(np.linalg.inv(np.matmul(np.transpose(x1),x1)),np.
 ↪transpose(x1)),y)
```

1.1 一元线性回归模型

以父亲和母亲的平均身高为因变量, 孩子身高为自变量. 一元线性回归，我们从数据集中提取整理所需数据，用矩阵 X 和向量 Y 表示。

[3]:
```
def read_sig(data):
    x = []
    y = []
    for line in data.values:
        x.append([(line[1]+line[2])/2])
        y.append([line[7]])
    x = np.array(x)
    y = np.array(y)
    return x,y
```

我们用自己定义的 lg 函数做最小二乘计算，训练结果如下所示

[4]:
```
x,y = read_sig(data)
lg(x,y)
```

[4]:
```
array([[50.59048246],
       [ 0.71258499]])
```

我们使用 sklearn 库中的线性回归模型进行训练，训练结果如下所示，与 lg 函数的结果一致，说明 sklearn 也是通过最小二乘实现的线性回归。

[5]:
```
x,y = read_sig(data)

from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(x,y)
(linreg.intercept_,linreg.coef_)
```

[5]:
```
(array([50.59048246]), array([[0.71258499]]))
```

1.2 多元线性回归模型

以父亲和母亲的身高为因变量, 孩子身高为自变量, 多元线性回归我们从数据集中提取整理所需数据，用矩阵 X 和向量 Y 表示。

[6]:
```
def read_multi(data):
    x = []
    y = []
    for line in data.values:
```

```
        x.append([line[1],line[2]])
        y.append([line[7]])
    x = np.array(x)
    y = np.array(y)
    return x,y
```

下面我们统一使用 sklearn 提供的线性回归模型进行计算.

[7]:
```
x,y = read_multi(data)

from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(x,y)
print (linreg.intercept_)
print (linreg.coef_)
```

```
[50.60937532]
[[0.40870579 0.30378638]]
```

2 模型测试

我们使用交叉验证的方法进行模型测试，考虑到性别差异，我们采用按孩子性别进行分层抽样的方法将数据集分成 10 折，每次测试中取一折作为测试集，其余为训练集。为此我们首先读取孩子的性别数据。并利用 sklearn 中的 StratifiedKFold 工具定义数据集和训练集的划分方法。

[8]:
```
def read_sex(data):
    sex = []
    for line in data.values:
        if line[6]=='male':
            sex.append([1])
        else:
            sex.append([0])
    return np.array(sex)

folds = 10
from sklearn.model_selection import KFold,StratifiedKFold
from sklearn import metrics
```

```
skf = StratifiedKFold(n_splits=folds)
```

我们采用均方误差度量模型的性能，取交叉验证结果的平均为最终度量.

2.1 一元回归模型测试

[9]:
```
x,y = read_sig(data)
sex = read_sex(data)
MSE = 0

for i,(train, test) in enumerate(skf.split(x,sex)):
    linreg = LinearRegression()
    linreg.fit(x[train],y[train])
    pred = linreg.predict(x[test])
    fold_MSE = metrics.mean_squared_error(y[test], pred)
    MSE += fold_MSE
    print ("Fold",i+1," \tTrain Results",linreg.intercept_,linreg.
 ↪coef_,"\n\t\tMSE",fold_MSE)
print ("Average MSE\t",MSE/folds)
```

```
Fold 1          Train Results [54.87321111] [[0.68788739]]
                MSE 32.08531000285515
Fold 2          Train Results [56.59873284] [[0.67729092]]
                MSE 33.36581065224919
Fold 3          Train Results [49.36265849] [[0.7193495]]
                MSE 37.06653369388921
Fold 4          Train Results [48.95521889] [[0.72256327]]
                MSE 30.665286764480996
Fold 5          Train Results [48.56435484] [[0.72376122]]
                MSE 29.99488031776479
Fold 6          Train Results [48.7321496] [[0.7230082]]
                MSE 35.586173408651575
Fold 7          Train Results [48.17726268] [[0.7268352]]
                MSE 34.01000014871627
Fold 8          Train Results [58.07528184] [[0.67058853]]
                MSE 27.61441685928544
Fold 9          Train Results [47.26005119] [[0.73180736]]
```

```
                MSE 34.73612619617521
Fold 10         Train Results [47.86280189] [[0.72802635]]
                MSE 32.284357932040685
Average MSE      32.74088959761086
```

一元回归模型的 MSE 为 32.74089

2.2 多元回归模型测试

[10]:
```
x,y = read_multi(data)
sex = read_sex(data)
MSE = 0

for i,(train, test) in enumerate(skf.split(x,sex)):
    linreg = LinearRegression()
    linreg.fit(x[train],y[train])
    pred = linreg.predict(x[test])
    fold_MSE = metrics.mean_squared_error(y[test], pred)
    MSE += fold_MSE
    print ("Fold",i+1," \tTrain Results",linreg.intercept_,linreg.
 ↪coef_,"\n\t\tMSE",fold_MSE)
print ("Average MSE\t",MSE/folds)
```

```
Fold 1          Train Results [51.20195173] [[0.41869314 0.29056237]]
                MSE 32.28220908244237
Fold 2          Train Results [56.44223542] [[0.38114802 0.29715764]]
                MSE 32.53360464996961
Fold 3          Train Results [50.06117421] [[0.40708329 0.30837293]]
                MSE 36.66454798222209
Fold 4          Train Results [48.96653317] [[0.41630987 0.30629129]]
                MSE 30.86922887715666
Fold 5          Train Results [50.11684585] [[0.40909643 0.30583143]]
                MSE 29.657287548743895
Fold 6          Train Results [49.43459854] [[0.40886185 0.31023668]]
                MSE 35.252849608945695
Fold 7          Train Results [48.46772535] [[0.40427674 0.32084109]]
                MSE 33.24667534186874
Fold 8          Train Results [59.43480636] [[0.40894633 0.25392269]]
```

```
                MSE 30.301054858945395
Fold 9          Train Results [46.86823118] [[0.40343253 0.33042938]]
                MSE 33.72419221113123
Fold 10         Train Results [44.61899301] [[0.43567309 0.31040058]]
                MSE 32.842535943227396
Average MSE      32.73741861046531
```

多元回归模型的 MSE 为 32.73742

说明在当前数据集下, 两种模型的表现不相上下。