朱金秋 1220086621
理学院 应用统计

作业1 牛顿法

In [206]:

```
def sigmoid(x, Θ_1, Θ_2):
    z = (Θ_1*x + Θ_2).astype("float_")
    return np.exp(z) / (1.0 + np.exp(z))
def log_likelihood(x, y, Θ_1, Θ_2):
    sigmoid_probs = sigmoid(x, Θ_1, Θ_2)
    return np.sum(y * np.log(sigmoid_probs) + (1 - y) * np.log(1 - sigmoid_probs))
def gradient(x, y, Θ_1, Θ_2):
    sigmoid_probs = sigmoid(x, Θ_1, Θ_2)
    return np.array([[np.sum((y - sigmoid_probs) * x),np.sum((y - sigmoid_probs) * 1)]])
def hessian(x, y, Θ_1, Θ_2):
    sigmoid_probs = sigmoid(x, Θ_1, Θ_2)
    d1 = np.sum((sigmoid_probs * (1 - sigmoid_probs)) * x * x)
    d2 = np.sum((sigmoid_probs * (1 - sigmoid_probs)) * x * 1)
    d3 = np.sum((sigmoid_probs * (1 - sigmoid_probs)) * 1 * 1)
    H = np.array([[d1, d2],[d2, d3]])
    return H
def newtons_method(x, y,Θ_1,Θ_2):
    """
    :param x (np.array(float)): Vector of Boston House Values in dollars
    :param y (np.array(boolean)): Vector of Bools indicting if house has > 2 bedrooms:
    :returns: np.array of logreg's parameters after convergence, [Θ_1, Θ_2]
    """

    # Initialize log_likelihood & parameters
     # The intercept term
    Δl = np.Infinity
    l = log_likelihood(x, y, Θ_1, Θ_2)
    # Convergence Conditions
    δ = .0000000001
    max_iterations = 50
    i = 0
    while abs(Δl) > δ and i < max_iterations:
        i += 1
        g = gradient(x, y, Θ_1, Θ_2)
        hess = hessian(x, y, Θ_1, Θ_2)
        H_inv = np.linalg.inv(hess)
        # @ is syntactic sugar for np.dot(H_inv, g.T)¹
        Δ = H_inv @ g.T
        ΔΘ_1 = Δ[0][0]
        ΔΘ_2 = Δ[1][0]

        # Perform our update step
        Θ_1 += ΔΘ_1
        Θ_2 += ΔΘ_2
        print('alpha_hat: ',Θ_2,'beta_hat:',Θ_1)
        # Update the log-likelihood at each iteration
        l_new = log_likelihood(x, y, Θ_1, Θ_2)
        Δl = l - l_new
        l = l_new
    return np.array([Θ_1, Θ_2])
```

初始值为（0.0,0.0）时

通过newton法，能在5步收敛到 alpha_hat: 3.8194399105149293 beta_hat: -0.0864829445827074

In [208]:

```
x = np.array([21,24,25,26,28,31,33,34,35,37,43,49,51,55,25,29,43,44,46,46,51,55,56,58])
y = np.array([1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0])
newtons_method(x, y, Θ_1 = 0.0, Θ_2 = 0.0)  #初始为(0,0)时
```

```
alpha_hat:  3.270078543985945 beta_hat: -0.0745840053499076
alpha_hat:  3.78812700250148 beta_hat: -0.08582023900921554
alpha_hat:  3.819323997504314 beta_hat: -0.08648051906752362
alpha_hat:  3.8194399088861823 beta_hat: -0.08648294454878022
alpha_hat:  3.819439910514929 beta_hat: -0.0864829445827074
```

Out[208]:

```
array([-0.08648294,  3.81943991])
```

初始值为0.1时

通过newton法，在初始值为(0.1,0.1)时无法收敛，因为在初始值过大时，例如exp(z)中的z= ax+b过大，会造成极大溢出，程序无法进行迭代，而造成无法收敛

In [211]:

```
x = np.array([21,24,25,26,28,31,33,34,35,37,43,49,51,55,25,29,43,44,46,46,51,55,56,58])
y = np.array([1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0])
newtons_method(x, y, Θ_1 = 0.4, Θ_2 = 0.4)#初始为(0.1,0.1)时
```

```
alpha_hat:  2434270.7675871057 beta_hat: -106285.80703185187

<ipython-input-206-cff1cb858104>:3: RuntimeWarning: overflow encountered in exp
  return np.exp(z) / (1.0 + np.exp(z))
<ipython-input-206-cff1cb858104>:3: RuntimeWarning: invalid value encountered in tru
e_divide
  return np.exp(z) / (1.0 + np.exp(z))
<ipython-input-206-cff1cb858104>:6: RuntimeWarning: divide by zero encountered in lo
g
  return np.sum(y * np.log(sigmoid_probs) + (1 - y) * np.log(1 - sigmoid_probs))
<ipython-input-206-cff1cb858104>:6: RuntimeWarning: invalid value encountered in mul
tiply
  return np.sum(y * np.log(sigmoid_probs) + (1 - y) * np.log(1 - sigmoid_probs))
```

Out[211]:

```
array([-106285.80703185,  2434270.76758711])
```

作业2 robust regression -t

In [108]:

```
###稳健回归
school = np.array([12,13,19,16,8,14,39,23,29,72,67,3,61,66,29,38,111,66,13,68,68,36,16,28,52,63,49,7
data = school.reshape((-1,7))
```

LS

In [109]:

```
data[:,1]
```

Out[109]:

```
array([ 13,  29,  38,  16,  40,  14,  44,  60,  16,  37,  20,  11,  29,
        14,  38,  27,  32,  56,  32,  42,  30,  18,  41,  23, 111,  26,
        16,  38,  19,  16,  13,  23,  32,  21,  28,  22,  20,  32,  26,
        40,  19,  17,  29,  27,  26,  36])
```

In [131]:

```
##beta_head = (x'x)^-1*x'y
X = data[:,1:]
y = data[:,0]
a = np.dot(np.linalg.inv(np.dot(X.T,X)),X.T)
beta_hat_LS = np.dot(a,y)
beta_hat_LS
```

Out[131]:

```
array([ 0.23752321, -0.0112    ,  0.08387168,  0.23303386,  0.02188319,
        0.14832735])
```

In [79]:

```
y_pre = np.dot(X,beta_hat_LS)
y_pre
```

Out[79]:

```
array([12.17235085, 23.52475702, 27.92196996, 34.72605961, 41.21069548,
       27.85840406, 39.11322716, 44.64100549, 27.14806422, 22.90374835,
       20.7416183 ,  7.09924428, 30.81474046, 10.144433  , 29.2819321 ,
       38.85894839, 26.90280114, 39.53559759, 28.39386839, 28.81831336,
       24.62212059, 14.25381514, 26.87580426, 21.4411763 , 53.08245625,
       30.80322238, 21.2891753 , 31.01998696, 19.58970092, 30.39969863,
       22.15214086, 19.19328053, 29.80244419, 33.60746219, 31.51118819,
       27.0245775 , 16.39177486, 19.33693354, 42.88001457, 26.0647608 ,
       19.25523598, 14.97930442, 29.39400289, 23.13053963, 30.05287656,
       44.35906881])
```

robust regression program using t-distribution based-weight

In [213]:

```
def IRLS(y, X, maxiter, tolerance=0.001):
    n, p = X.shape
    B = np.repeat(0, p)#初始B
    sigma = 1#初始sigma
    # W是对角线上为w的对角矩阵
    w = 1+1/(1+(y-np.dot(X,B)/np.sqrt(sigma))**2)  #t-分布w的估计公式（文件），自由度为1
    W = np.diag(w)
    #z = np.linalg.inv(W).dot(y)
    B = np.dot(np.linalg.inv(X.T.dot(W).dot(X)),(X.T.dot(W).dot(y)))  #B的估计公式
    sigma = np.mean(w*((y-np.dot(X,B))**2))     #sigma的估计公式
    for _ in range(maxiter):   #迭代这么多次
        _B = B
        _w = w
        _sigma = sigma
        _W = np.diag(_w)
        B = np.dot(np.linalg.inv(X.T.dot(_W).dot(X)),(X.T.dot(_W).dot(y)))  #更新
        sigma = np.mean(_w*((y-np.dot(X,_B))**2))  #更新
        w = 1+1/(1+(y-np.dot(X,B)/np.sqrt(sigma))**2)  #更新
        tol = sum(abs(B - _B))  #容忍度
        print("Tolerance = %s" % tol)
        if tol < tolerance:
            return B,w,sigma
    return B,w
```

In [214]:

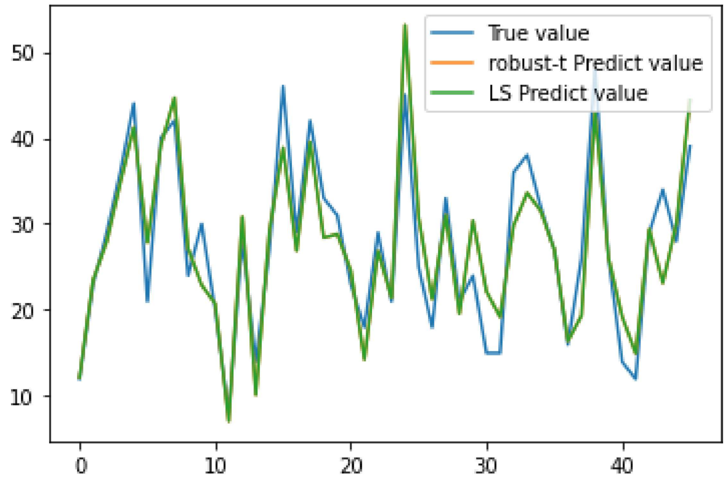
```
B_IRLS,weights,sigma = IRLS(y, X, maxiter=10000, tolerance=0.001)
```

```
Tolerance = 0.0
```

robust和LS的对比

In [215]:

```
from sklearn.metrics import mean_squared_error
plt.figure()
plt.plot(np.arange(len(y)), y, label="True value")
plt.plot(np.arange(len(y)), np.dot(X,B_IRLS),  label="robust-t Predict value")
plt.plot(np.arange(len(y)), y_pre, label="LS Predict value")
plt.legend()
plt.show()
print('robust-t Predict value MSE:',mean_squared_error( np.dot(X,B_IRLS),y))
print('LS Predict value MSE:',mean_squared_error(y_pre,y))
```




```
robust-t Predict value MSE: 17.52114282199999
LS Predict value MSE: 17.52113191297699
```

可以看出两者拟合结果相差很小

输出权重

In [193]:

```
weights
```

Out[193]:

```
array([1.00776167, 1.00212701, 1.00132933, 1.00086321, 1.00057608,
       1.00264646, 1.00070044, 1.00064193, 1.0019796 , 1.001213  ,
       1.00281589, 1.01703788, 1.00144978, 1.00552018, 1.00155594,
       1.00052132, 1.00132372, 1.00063257, 1.00101435, 1.00115889,
       1.00213931, 1.00337356, 1.00132357, 1.00254976, 1.00056725,
       1.00184727, 1.00353594, 1.0010241 , 1.00252282, 1.00201262,
       1.00527041, 1.00514426, 1.00084916, 1.00076737, 1.00109491,
       1.00154026, 1.00438626, 1.0016105 , 1.00048159, 1.00166118,
       1.00597042, 1.00798404, 1.00133748, 1.00093534, 1.00144499,
       1.00075117])
```

In [198]:

```
np.argmin((weights))#最小的那个学校
```

Out[198]:

38

In [220]:

```
np.argmax((y-np.dot(X,B_IRLS)/np.sqrt(sigma))**2)
```

Out[220]:

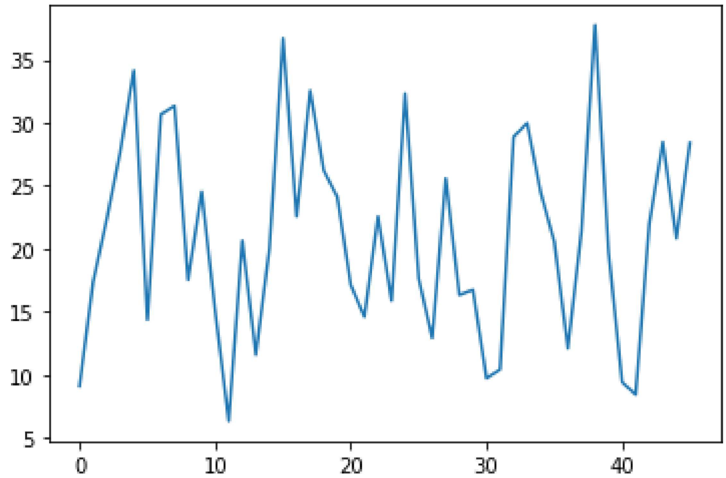
38

In [228]:

```
plt.plot((y-np.dot(X,B_IRLS)/np.sqrt(sigma)))
```

Out[228]:

[<matplotlib.lines.Line2D at 0x2651b318700>]



第37个学校的权重最小（python从0开始计数）

根据权重公式是因为那个学校ei/sigma最大。在t分布中为分布的中心

作业3

Write out the code to estimate the parameters _ and jj using the LS-method, with the following sample covariance matrix.

$$S = \begin{pmatrix} 1.895 & 0.908 & 0.926 \\ 0.908 & 2.367 & 0.718 \\ 0.926 & 0.718 & 1.984 \end{pmatrix}$$

$$\boldsymbol{\Lambda} = \boldsymbol{\lambda} = \lambda \mathbf{1} = \lambda(1, 1, \ldots, 1)',$$

$$\boldsymbol{\Sigma}(\boldsymbol{\theta}) = \lambda^2 \mathbf{1}\mathbf{1}' + \boldsymbol{\Psi},$$

$$F_{LS}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{p}[s_{jk} - \sigma_{jk}(\boldsymbol{\theta})]^2$$

$$\dot{F}_{LS\lambda}(\theta) = -(2\lambda)\mathbf{1}'\left[S - \sum(\theta)\right]\mathbf{1} = 0$$

$$-(2\lambda)(11.35 - 9\lambda^2 - \Psi_{11} - \Psi_{22} - \Psi_{33}) = 0$$

$$\dot{F}_{LS\Psi_{jj}}(\theta) = -[s_{jj} - \sigma_{jk}(\theta)] = 0$$

$$1.895 - (\lambda^2 + \Psi_{11}) = 0$$
$$2.367 - (\lambda^2 + \Psi_{22}) = 0$$
$$1.984 - (\lambda^2 + \Psi_{33}) = 0$$

$\lambda^2 = 0.8506667 \Psi_{11} = 1.0443333 \Psi_{22} = 1.5163333 \Psi_{33} = 1.1333333$

代码:

```
> f=matrix(c(9,1,1,1,
+           1,1,0,0,
+           1,0,1,0,
+           1,0,0,1),4,4)
> rf=matrix(c(11.35,
+           1.895,
+           2.367,
+           1.984),c(4,1))
> solve(f,rf)
          [,1]
[1,] 0.8506667
[2,] 1.0443333
[3,] 1.5163333
[4,] 1.1333333
```

得出估计值