**SD211 TP Logistic Regression**

**WEI CHEN**

**weichen@whu.edu.cn**

# **Import**

**import** **cervicalcancerutils** **as** **cancer**

**import** **numpy** **as** **np**

**from** **scipy.optimize** **import** check\_grad, line\_search

**import** **matplotlib.pyplot** **as** **plt**

**from** **pywt** **import** threshold

filename = 'riskfactorscervicalcancer.csv'

X, y=cancer.load\_cervical\_cancer(filename)

n,p=X.shape

# **Tikhonov Regularisation**

**Question 1.1 Gradient, Hessian Matrix**

Note , and , then we can rewrite the objective function as

And is twice differentiable, and we calculate its Hessian matrix

where and

This matrix is definitely positive, so is convex.

**def** tikhonov(X, y, w0w):

val=0

grad\_w=0

grad2\_w=np.zeros((p+1,p+1))

**for** i in **range**(n):

x=np.insert(X[i,:],0,1).reshape(p+1,1)

s=np.ones(p+1)

s[0]=0

Is=np.diag(s)

z = np.exp(-y[i]\*x.T.dot(w0w)[0])

val=val+1/n\*np.log(1+z)+1/n\*rho/2\*np.sum((s\*w0w)\*\*2)

grad\_w=grad\_w+1/n\*(-y[i]\*x.T[0])\*z/(1+z)+1/n\*rho\*(s\*w0w)

grad2\_w=grad2\_w+1/n\*y[i]\*\*2 \* (x.dot(x.T)) \* z / (1+z)\*\*2 + 1/n\*rho\*Is

**return** val, grad\_w, grad2\_w

**Question 1.2 check\_grad**

We want to verify the function defined above is correct, with the help of the function scipy.optimize.check\_grad.

**def** f1(w0w):

**return** tikhonov(X, y, w0w)[0]

**def** grad(w0w):

**return** tikhonov(X, y, w0w)[1]

rho=1/n

w0w=np.random.rand(p+1)

print(check\_grad(f1, grad, w0w, epsilon=1e-6))

This function will return the 2-norm of the difference between grad(w0w) and the finite difference approximation of *grad* using f1. The result we obtain is

1.60841462877e-07

This difference is small enough, which indicates the gradient of w calculated by function grad is verified.

**Question 1.3 Newton Method**

|  |
| --- |
|  |

We fix the initial point as , and propose the stop condition as .

**def** newton(w0w,epsilon):

gradient=tikhonov(X, y, w0w)[1]

G=list([])

**while** np.sqrt(np.sum(gradient\*\*2))>=epsilon:

gamma=np.linalg.inv(tikhonov(X, y, w0w)[2])

w0w=w0w-gamma.dot(gradient)

gradient=tikhonov(X, y, w0w)[1]

G.append(np.log10(np.sqrt(np.sum(gradient\*\*2))))

**return** w0w, G

epsilon=1e-10

rho=1/n

w0w=np.zeros(p+1)

w0w\_opt,G=newton(w0w,epsilon)

plt.figure()

plt.plot(G,'--o')

plt.xlabel('iteration')

plt.ylabel('logarithmic 2-norm of grad\_w')

plt.grid()

plt.show()

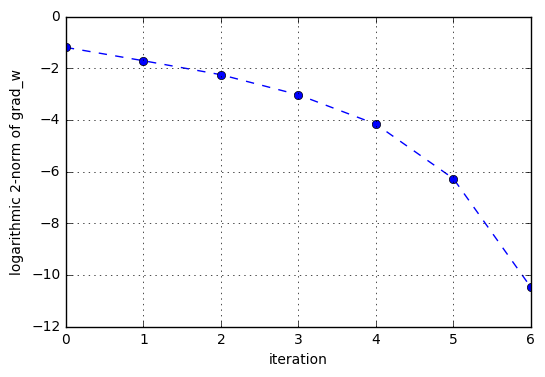


Figure. Evolution of

**Question 1.4 Newton Method**

If we take the initial point as , we will find the algorithm doesn’t converge.

Because one of the condition to ensure the convergence of Newton method is that the initial point should be chosen close enough to a local minimum .

We verify that

So we conclude that is too far from the minimum , which doesn’t ensure the convergence.

**Question 1.5 Line Search**

The idea of line search is to choose the pace adaptively using local information. To apply the line search to Newton method, we take

|  |
| --- |
|  |

In Armijo’s line search, we choose the pace length in each step as

where and l is the smallest nonnegative integer such that

Basically, the larger is, the larger will be; b determines the maximum pace length; controls the number of levels of pace length. The following two figures show the roles of and *a*.

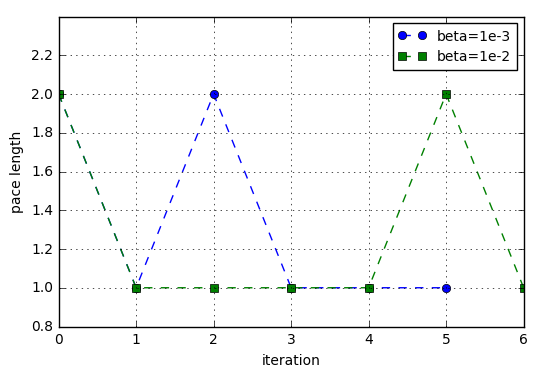
****

Figure. The larger is, the more possible is selected as large value

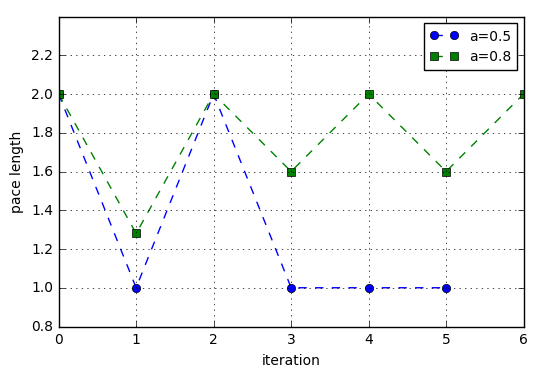
****

Figure. The smaller is, the more possible is selected as large value

In order to let the algorithm converge as fast as possible, we select .

**def** linesearch(f,grad,w0w):

a=0.5

b=2

beta=1e-3

l=0

direction=np.linalg.inv(tikhonov(X, y, w0w)[2]).dot(tikhonov(X, y, w0w)[1])

w0wplus=w0w-b\*a\*\*l\*direction

**while** f(w0wplus)>f(w0w)+beta\*direction.dot(w0wplus-w0w):

l=l+1

w0wplus=w0w-b\*a\*\*l\*direction

gamma=b\*a\*\*l

**return** gamma

epsilon=1e-10

w0w=np.zeros(p+1)

gradient=grad(w0w)

G=list([])

**while** np.sqrt(np.sum(gradient\*\*2))>=epsilon:

gamma=linesearch(f1,grad,w0w)

direction=np.linalg.inv(tikhonov(X, y, w0w)[2]).dot(gradient)

w0w=w0w-gamma\*direction

gradient=grad(w0w)

G.append(np.log10(np.sqrt(np.sum(gradient\*\*2))))

w0w\_opt=w0w

plt.figure()

plt.plot(G,'--o')

plt.xlabel('iteration')

plt.ylabel('logarithmic 2-norm of grad\_w')

plt.grid()

plt.show()

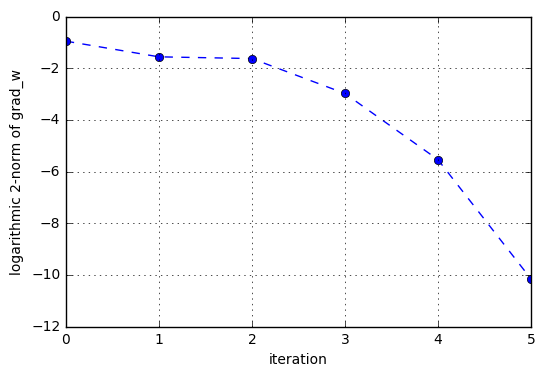


Figure. Evolution of with .

converges to in 5 iterations.

# **Sparsity Regularisation**

**Question 2.1 Gradient, Hessian Matrix**

where

Actually, is not differentiable at the point , so we calculate its sub gradient instead.

Newton method requires that the objective function is three times continuously differentiable, however we can see that is not differentiable at .So Newton method cannot be applied to this case.

**Question 2.2 Proximal Gradient Method**

Rewrite the objective function as

where

and

The proximal gradient method is given by

|  |
| --- |
|  |

Where is called **soft thresholding function** which is defined by

and the gradient of

The Hessian matrix of

We have proved that this matrix is positive definite, so the objective function is convex.

**def** sparse(X, y, w0w):

val=0

grad\_w=0

**for** i in range(n):

x=np.insert(X[i,:],0,1).reshape(p+1,1)

s=np.ones(p+1)

s[0]=0

z = np.exp(-y[i]\*x.T.dot(w0w)[0])

val=val+1/n\*np.log(1+z)

grad\_w=grad\_w+1/n\*(-y[i]\*x.T[0])\*z/(1+z)

**return** val, grad\_w

**Question 2.3 Proximal Gradient with Line Search**

When the divergence between the objective values in two iterations is small enough, we assume the algorithm converges. And the principle of the algorithm is as follows:

|  |
| --- |
| Initialise    , |
| repeat   * choose |
| until |

**def** f2(w0w):

**return** sparse(X, y, w0w)[0]

**def** F2(w0w):

**return** sparse(X, y, w0w)[0]+rho\*np.linalg.norm(w0w[1:],1)

**def** grad(w0w):

**return** sparse(X, y, w0w)[1]

**def** proximal(w0w,epsilon):

gradient=grad(w0w)

C=list([])

k=0

**while** k<2 or np.abs(C[-1]-C[-2])>=epsilon:

gamma=line\_search(f2, grad, w0w, -grad(w0w) )[0]

w0w[0]=(w0w-gamma\*gradient)[0]

w0w[1:]=threshold(w0w-gamma\*gradient,rho\*gamma, mode='soft')[1:]

gradient=grad(w0w)

C.append(F2(w0w))

k=k+1

**return** w0w, C

epsilon=1e-6

w0w=np.zeros(p+1)

rho=0.1

w0w\_opt,C=proximal(w0w,epsilon)

plt.figure()

plt.plot(C,'--o')

plt.xlabel('iteration')

plt.ylabel('objective value')

plt.grid()

plt.show()

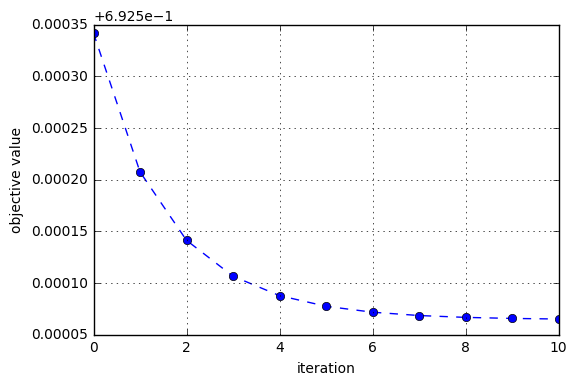
****

Figure. Evolution of with .

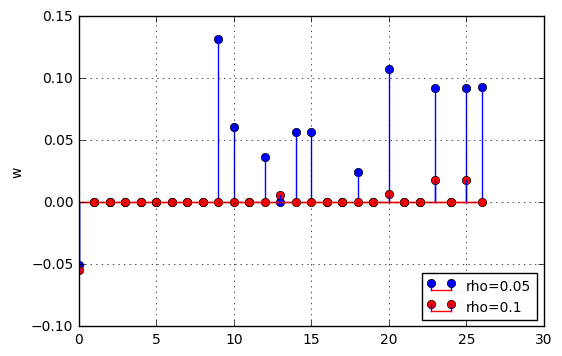


Figure. Solution of with different sparsity level

We find that -regularization will lead to sparse solution of . the sparsity level is determined by . Larger will lead to sparser .

# **Comparison**

**Question 3.1 Property of the two optimisation problems**

* **Sparsity promotion**

The main difference between and regularization is that can yield sparse models while doesn't. Sparse model is a great property to have when dealing with high-dimensional data.

* **Solution of the optimization problem**

-regularized objective function is smooth. The solution that minimizes the objective function is the stationary point (0-derivative point).

However, -regularized objective function is non-smooth. It's not differentiable at 0. The optimal solution of a function is either the point with 0-derivative or one of the irregularities (corners, kinks, etc.). So, it's possible that the optimal solution is 0 even if 0 isn't the stationary point.

* **Prediction error**

Typically regularization is much better for minimizing prediction error rather than regularization. Because when two predictors are highly correlated, regularization will simply pick one of the two predictors. In contrast, the regularization will keep both of them and jointly shrink the corresponding coefficients a little bit.

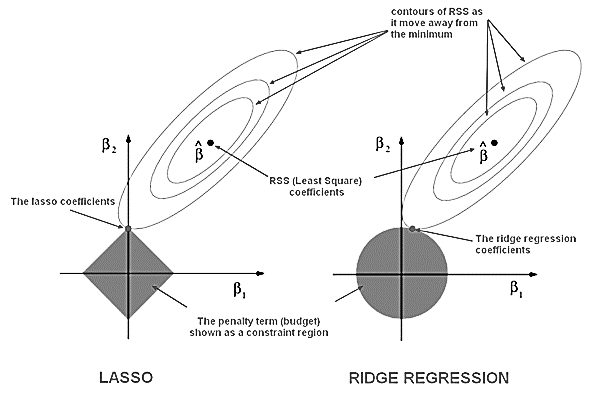


Figure. regularization leads to sparse solution, while regularization doesn’t.

**Question 3.2 Solution resulted from two types of regularisation**

We fix the regularisation parameter the same in two objective functions: **.** Then we obtain the estimated of two problems with different regularisations.

]:

plt.stem(r1,markerfmt='bo',label='Tikhonov regularisation')

plt.stem(r2,markerfmt='ro',label='Sparsity regularisation')

plt.grid()

plt.legend(loc=2,fontsize=10)

plt.ylabel('w')

plt.show()

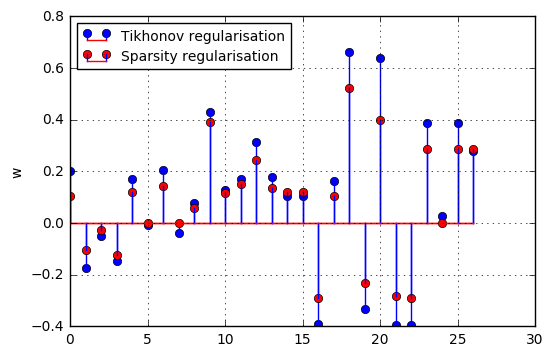


Figure. Solution of of two optimisation problems

In logistic regression,

So we compute the probability based on we estimated from the two problems.

y\_est1=np.zeros(n)

y\_est2=np.zeros(n)

**for** i in range(n):

x=np.insert(X[i,:],0,1).reshape(p+1,1)

y\_est1[i]=1/(1+np.exp(-x.T.dot(r1)))

y\_est2[i]=1/(1+np.exp(-x.T.dot(r2)))

y\_est0=y\*0.5+0.5

plt.figure()

plt.scatter(range(n),y\_est0,c='black')

plt.scatter(range(n),y\_est1,label='Tikhonov regularisation')

plt.scatter(range(n),y\_est2,c='r',label='Sparsity regularisation')

plt.axhline(y=0.5,linestyle='--',color='black')

plt.grid()

plt.xlabel('i')

plt.ylabel('p(y=1|X,w)')

plt.legend(loc=1,fontsize=10)

plt.show()

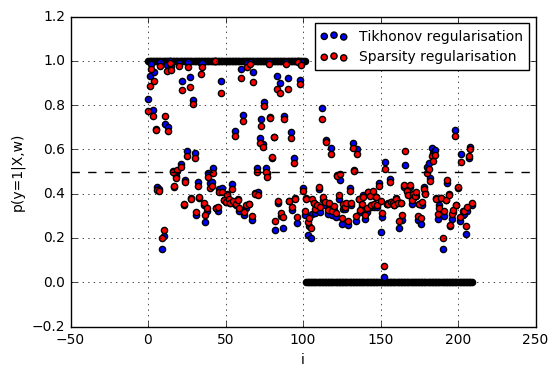


Figure. Estimated of two optimisation problems

We can conclude that both of this two regularizations will lead to a lot of prediction errors. Regarding this, it is very possible that the data set is not linear separable.