CS289–Spring 2017 — Homework 4 Solutions

Shuhui Huang, SID 3032129712

1. Logistic Regression with Newtons Method

(a) As $s_i = s(X_i \cdot w)$, the derivative of cost function is :
$ds(\gamma) = s(\gamma)(1 - s(\gamma))$
$\frac{\partial J}{\partial w} = 2\lambda w - \sum(\frac{y_i}{s_i} - \frac{1-y_i}{1-s_i})ds_i$
$= 2\lambda w - \sum(\frac{y_i}{s_i} - \frac{1-y_i}{1-s_i})(s_i)(1-s_i)X_i = 2\lambda w - \sum(y_i - x_i)X_i = 2\lambda w - X^T(y-s)$
So the gradient is : $2\lambda w - X^T(y-s)$

(b) The Hessian matrix of J(w) should be:
$\frac{\partial^2 J(w)}{\partial w^2} = \frac{\partial}{\partial w}(\frac{\partial J}{\partial w}) = -\sum_{i=1}^n s_i(1-s_i)X_i X_i^T + 2\lambda I_d = X^T\Omega X + 2\lambda I_d$, where

$$\Omega = \begin{bmatrix} s_1(1-s_1) & 0 & \cdots & 0 \\ 0 & s_2(1-s_2) & \cdots & 0 \\ \cdots & & & \\ 0 & 0 & \dots & s_n(1-s_n) \end{bmatrix}$$

(c) one iteration of Newtons method for this problem is :
Solve for e in normal equations: $(X^T\Omega X + 2\lambda I_d)e = X^T(y-s) - 2\lambda w$,
$e = (X^T\Omega X + 2\lambda I_d)^{-1}(X^T(y-s) - 2\lambda w)$
$w \leftarrow w + e$

(d) (a).$s^{(0)} = s(X \cdot w^{(0)}) = s(3, 1, 1, -1) = \begin{bmatrix} 0.9526 \\ 0.7311 \\ 0.7311 \\ 0.2689 \end{bmatrix}$

(b).$w \leftarrow w + e$, $e = (X^T\Omega X + 2\lambda I_d)^{-1}(X^T(y-s) - 2\lambda w)$
we get the $w^{(1)} = \begin{bmatrix} 0.0687 \\ 2.1089 \\ -4.3109 \end{bmatrix}$

(c).$s^{(1)} = s(X \cdot w^{(1)}) = s(11.8, 6.4, 1.3, -4.1) = \begin{bmatrix} 0.8824 \\ 0.8894 \\ 0.0996 \\ 0.1059 \end{bmatrix}$

(d). $w^{(2)} \leftarrow w^{(1)} + e^{(1)}$ so we get the $w^{(2)} = \begin{bmatrix} 0.0058 \\ 2.3696 \\ 0.4432 \end{bmatrix}$

2.l_1 and l_2-Regularization

(a) $J(w) = |Xw - y|^2 + \lambda||w||_{l_1} = \sum_{i=1}^{n}(\sum_{j=1}^{d} x_{ij}w_j - y_i)^2 + \lambda\sum_{j=1}^{d}|w_j|$
$= y^2 + \sum_{j=1}^{d}[(x_{*j}w_j)^2 - 2y\cdot(x_{*j}w_j) + \lambda|w_j|]$

(b) if $w_i^* > 0$, the value of w_i^* is:
let the cost function minimize, the derivative of it is : $\frac{\partial J(w)}{\partial w} = 2X^TXdw - 2X^Ty + \lambda = 0$
we get the $\hat{w} = (X^TX)^{-1}(X^Ty - \frac{1}{2}\lambda) = \frac{1}{n}(X^Ty - \frac{1}{2}\lambda)$ so $w_i^* = \frac{1}{n}\sum_{j=1}^{n}(X_{ji}\cdot y_i - \frac{1}{2}\lambda)$

(c) if $w_i^* < 0$, the derivative of it is : $\frac{\partial J(w)}{\partial w} = 2X^TXdw - 2X^Ty - \lambda = 0$
so $\hat{w_i^*} = \frac{1}{n}(X^Ty + \frac{1}{2}\lambda) = \frac{1}{n}\sum_{j=1}^{n}(X_{ji}\cdot y_i + \frac{1}{2}\lambda)$

(d) In oder to have $w_i^* = 0$, we should have $|X^Ty| \leq \frac{1}{2}\lambda$

(e) if we use ridge regression, the f function will be:
$[(x_{*j}w_j)^2 - 2y\cdot(x_{*j}w_j) + \lambda|w_j|^2]$
to minimize the f function, the derivative will be:
$(X^TX + \lambda)w = X^Ty$
$\therefore \hat{w} = (X^TX + \lambda)^{-1}X^Ty$ so the new condition for $x_i^* = 0$ is:
$\lambda = \infty$ or $X^Ty = 0$.
it is different from part (4) for the condition is X,y independent of λ while (4) dependent on λ.

3.Regression and Dual Solutions

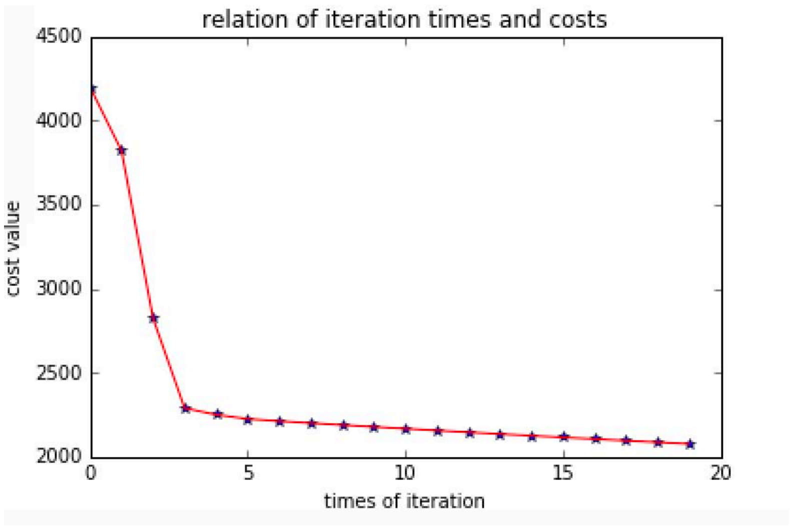
(a) $\nabla |w|^4 = \nabla ||w||_2^4 = \nabla(\sum_{i=1}^d w_i^2)^2 = 4(\sum_{i=1}^d w_i)(\sum_{i=1}^d w_i^2)$
And $\nabla |Xw-y|^4 = 4||Xw-y||_2^2 X^T(Xw-y)$

(b) The derivative of cost function is :
$\nabla J_w = 4||Xw-y||_2^2 X^T(Xw-y) + 2\lambda w$
And $\nabla^2 J_w = 12X^TX(Xw-y)^2 + 2\lambda > 0$ So there is only one w that satisfy $\nabla J_w = 0$, which is the optimum w^*
So to find w^* equals to find w satisfies $\nabla J_w = 4||Xw-y||_2^2 X^T(Xw-y) + 2\lambda w$. So the value of w^* is:
$w^* = (2||Xw^*-y||_2^2 X^TX + \lambda)^{-1} \cdot (2||Xw^*-y||_2^2 X^T)y$
Therefore, $a = (X^T)^{-1}w^* = (2||Xw^*-y||_2^2 X^TX + \lambda)^{-1} \cdot (2||Xw^*-y||_2^2)y$

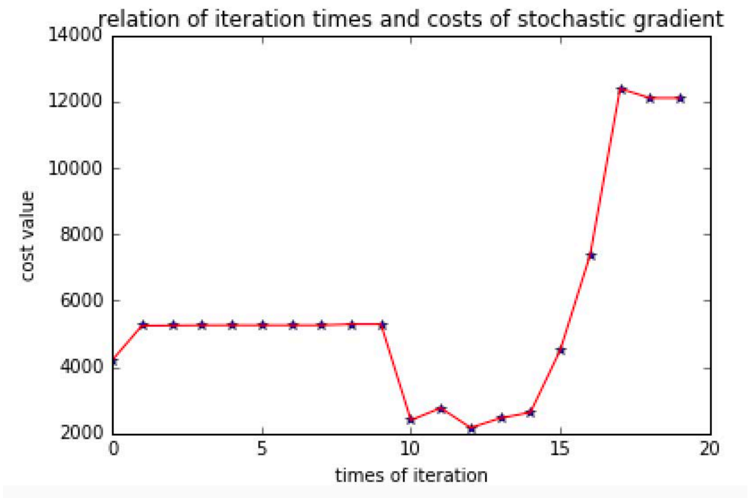
(c) First, prove the optimal solution has the form $w^* = \sum a_iX_i = X^Ta$:
the derivative of loss function is $\nabla_w L(w^Tx, y) + \lambda w^2 = X^T \nabla L(w^Tx, y) + 2\lambda w = 0$
So $w^* = \frac{-\nabla(w^Tx,y)}{2\lambda}x = a^Tx$
When the loss function is convex, for $L(\theta x + (1-\theta y)) \leq \theta L(x) + (1-\theta)L(y), \quad (0 \leq \theta \leq 1)$
But if the loss function is not convex, the optimal solution still always have the form $w^* = \sum a_iX_i = X^Ta$, since this form represent all the local minimum and the global minimum.

4.Franzia Classification + Logistic Regression = Party!

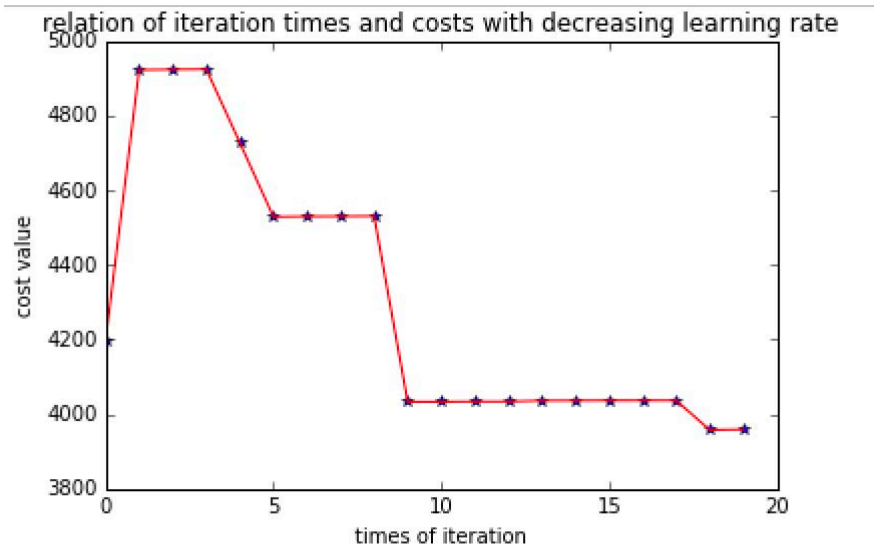
(a) batch gradient descent update equation for logistic regression with l_2 regularization is:
use error over entire training set:
the cost function is : $J(w) = \lambda|w|^2 - \sum_{i=1}^{n}(y_i Ins_i + (1-y_i)In(1-s_i))$
The derivative is $\nabla_w J = 2\lambda w - X^T(y-s)$ so the update is : $w \to w - \epsilon(2\lambda w - X^T(y-s))$ let regularization parameter value $\lambda = 0.05$, learning rate $\epsilon = \frac{1}{6000000}$.
$w \to w - \frac{1}{6000000}(0.1w - X^T(y-s)) = (1 - \frac{1}{60000000})w + \frac{1}{6000000} * X^T(y-s)$
And the relation between cost function and iteration times as plot:



(b) Stochastic gradient descent update equation for logistic regression with l_2 regularization is:
use error at a single sample of training set:
the cost function is : $J(w) = \lambda|w|^2 - \sum_{i=1}^{n}(y_i Ins_i + (1-y_i)In(1-s_i))$
The derivative is $\nabla_w J = 2\lambda w - X^T(y-s))$ so the update is : $w \to w - \epsilon(2\lambda w - (y_i - s_i)X_i)$
let regularization parameter value $\lambda = 0.05$, learning rate $\epsilon = \frac{1}{1000}$.
$w \to w - \frac{1}{1000}(0.1w - (y_i - s_i)X_i) = (1 - \frac{1}{10000})w + \frac{1}{1000} * ((y_i - s_i)X_i)$
And the relation between cost function and iteration times as plot:
And the difference is that plots of (a) is risk is strictly decreasing and converge. While the risk of (b) does not converge and oscillate within a band of values at the end.



(c) This is better as it solves the problem of the risk oscillating for a final set of values. The risk should now converge.
And the plot of training risk vs. number of iterations is :



(d) my kaggle score is : 0.97581

5.Real World Spam Classification

Daniel's mistake is that he stored the timestamp as the number of milliseconds since the previous midnight. If the spam spike was at a different time of day, because the magnitude of the time stamps would be close to others. However, because the spike is at midnight, the magnitude will have a huge difference as 11:59 PM is 86,340,000 ms while 12:00 AM is 0 ms. In this case, the linear decision boundary with no regularization and kernel will do poorly.

But we can come up with a method to fix this problem. we can add 30 minutes to each timestamp, mod 24 hours, then the magnitude of the time stamps would be close to others. In this way, identifying the spike become much simpler for a linear decision boundary. So we can use a linear SVM to get the result we want.

In [ ]:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 28 18:26:25 2017

@author: huangshuhui
"""
###problem1###

##4.a##
from numpy import *
x=mat([[0,3,1],[1,3,1],[0,1,1],[1,1,1]])
y=mat([[1,1,0,0]])
w0=mat([[-2,1,0]])
product=x*w0.T
s0=[]
lambda0=0.07
for i in range(0,4):
    s=1/(1+e**int(-product[i]))
    s0.append(s)

##4.b##
Omega=diag(s0)-diag(s0)*diag(s0)
e0=(x.T*Omega*x+mat(diag([0.14,0.14,0.14]))).I*(x.T*((y-s0).T)-2*lambda0 *(w0.T)
)
w1=w0.T+e0

##4.c##
product1=x*w1
s1=[]
for i in range(0,4):
    s=1/(1+e**float(-product1[[i]]))
    s1.append(s)

##4.d##
Omega1=diag(s1)-diag(s1)*diag(s1)
e1=(x.T*Omega1*x+mat(diag([0.14,0.14,0.14]))).I*(x.T*((y-s1).T)-2*lambda0 *(w1))
w2=w1+e1

###problem4###
import scipy.io as sio
import matplotlib.pyplot as plt

wines=sio.loadmat('/Users/huangshuhui/Desktop/study/CS289/hw2017/hw4/data.mat')
wines_test=wines['X_test']
wines_train=wines['X']
wines_trainlab=wines['y']

##4.a##
w0=mat([0.0001]*12)
```

```
x=mat(wines_train)

y=mat(wines_trainlab)
testx=mat(wines_test)

lambd=0.05
def batchgra(nums,x,y,w0,lambd):
    w=w0
    x=x
    y=y
    costnew=[]
    i=0
    while i<nums:
        product=x*w.T
        s=[]
        for j in range(0,len(x)):
            s_mid=1/(1+e**float(-product[j]))
            s.append(s_mid)
        ei=x.T*(y-mat(s).T)
        cost=lambd*w*w.T-(y.T*mat(log(s)).T+(1-y).T*(log(1-mat(s)).T))
        w=(1-0.001/len(x)*2*lambd)*w+0.001/len(x)*ei.T
        costnew.append(float(cost))
        i=i+1
    return costnew,w

cost_batch,w_batch=batchgra(20,x,y,w0)

plt.plot(cost_batch,'b*')
plt.plot(cost_batch,'r')
plt.xlabel('times of iteration')
plt.ylabel('cost value')
plt.title('relation of iteration times and costs')

##4.b##
import random
def stochgra(nums,x,y,w0):
    w=w0
    x=x
    y=y
    costnew=[]
    i=0
    while i<nums:
        sample=random.sample(range(0,len(x)),1)
        producti=x[sample]*w.T
        product=x*w.T
        si=1/(1+e**float(-producti))
        ei=(y[sample]-si)*x[sample]
        s=[]
        for j in range(0,len(x)):
            s_mid=1/(1+e**float(-product[j]))
            s.append(s_mid)
        cost=0.05*w*w.T-(y.T*mat(log(s)).T+(1-y).T*(log(1-mat(s)).T))
        w=(1-0.0001)*w+0.001*ei
        costnew.append(float(cost))
        i=i+1

    return costnew,w

cost_sto,w_sto=stochgra(20,x,y,w0)
plt.plot(cost_sto,'b*')
plt.plot(cost_sto,'r')
plt.xlabel('times of iteration')
plt.ylabel('cost value')
plt.title('relation of iteration times and costs of stochastic gradient')

##4.c##
def stochgra2(nums,x,y,w0):
    w=w0
    x=x
    y=y
    costnew=[]
    i=0
    while i<nums:
        sample=random.sample(range(0,len(x)),1)
        producti=x[sample]*w.T
        product=x*w.T
        si=1/(1+e**float(-producti))
        ei=(y[sample]-si)*x[sample]
        s=[]
        for j in range(0,len(x)):
            s_mid=1/(1+e**float(-product[j]))
            s.append(s_mid)
        cost=0.05*w*w.T-(y.T*mat(log(s)).T+(1-y).T*(log(1-mat(s)).T))
        w=(1-0.0001/(1+i))*w+0.001/(1+i)*ei
        costnew.append(float(cost))
        i=i+1
    return costnew,w

cost_sto2,w_sto2=stochgra2(20,x,y,w0)
plt.plot(cost_sto2,'b*')
plt.plot(cost_sto2,'r')
plt.xlabel('times of iteration')
plt.ylabel('cost value')
plt.title('relation of iteration times and costs with decreasing learning rate')

##4.d##
alpha=0.4
def logpredict(num,w0,x,y,testx,alpha,lambd):
    cost_batch,w_batch=batchgra(num,x,y,w0,lambd)
    wtx=w_batch*testx.T
    y_pre=[]
    for i in range(0,len(testx)):
        pi=(e**(alpha+float(wtx[...,i])))/(1+e**(alpha+float(wtx[...,i])))
        y_pre.append(int(pi+0.5))
    return y_pre

predict_y=logpredict(3000,w0,x,y,testx,alpha,lambd)
savetxt('wines.csv', predict_y, delimiter = ',')
```

```
#test the accuarcy#
count=0
y_pre1=logpredict(3000,w0,x,y,x,alpha,lambd)
for i in range(0,6000):
    if y_pre1[i]==y[i]:
        count+=1
count
```