CS 189 HW4 Yong-Chan Shin

$\boxed{Q1}$ (1) $J(w) = \lambda \|w\|_2^2 - \sum_{i=1}^{n} \left[y_i \ln s_i + (1-y_i) \ln(1-s_i) \right]$

$$\nabla J(w) = \lambda \cdot \frac{d}{dw} \sum_{k=1}^{n} w_k^2 - \sum_{i=1}^{n} \left(\frac{y_i}{s_i} \nabla s_i - \frac{1-y_i}{1-s_i} \nabla s_i \right)$$

$$= \lambda \cdot (2w) - \sum_{i=1}^{n} \left(\frac{y_i}{s_i} - \frac{1-y_i}{1-s_i} \right) \nabla s_i$$

$$= 2\lambda w - \sum_{i=1}^{n} \left(\frac{y_i}{s_i} - \frac{1-y_i}{1-s_i} \right) \cdot s_i (1-s_i) X_i \quad \left(\because s'(\gamma) = \frac{d}{d\gamma} \left(\frac{1}{1+e^{-\gamma}} \right) = \frac{e^{-\gamma}}{(1+e^{-\gamma})^2} = s(\gamma)(1-s(\gamma)) \right)$$

$$= 2\lambda w - \sum_{i=1}^{n} (y_i - s_i) X_i$$

$$= 2\lambda w - X^T (y - s)$$

(2) $\nabla^2 J(w) = \nabla_w \left(2\lambda w - X^T (y-s) \right)$

$$= 2\lambda \cdot I - \nabla (X^T y) + \nabla (X^T s)$$

$$= 2\lambda \cdot I + \sum_{i=1}^{d} s_i (1-s_i) \cdot X_i \cdot X_i^T \quad (\because \text{again, } s'(\gamma) = s(\gamma)(1-s(\gamma)))$$

$$= 2\lambda \cdot I + s^T (1-s) \cdot X^T \cdot X$$ where I is dxd matrix, 1 is dx1 matrix of 1

(3) $w \leftarrow w - (\nabla^2 J(w))^{-1} \nabla J(w)$

$\Leftrightarrow$ $w \leftarrow w - (2\lambda I + s^T (1-s) X^T \cdot X)^{-1} \cdot (2\lambda w - X^T (y-s))$

(4) (a) [0.95257413 0.73105858 0.73105858 0.26894142]

(b) [-1.85693596 1.45991964 -1.29914196]

(c) [0.95608609 0.77271069 0.54010806 0.15496768]

(d) [-1.75982464 1.83791738 -2.33758325

Q2 (1) $J(w) = |Xw - y|^2 + \lambda \|w\|_1$

$$= \sum_{i=1}^{n} \left((Xw)_i - y_i \right)^2 + \lambda \sum_{i=1}^{d} |w_i|$$

$$= \sum_{i=1}^{n} \left(x_i^T \cdot w - y_i \right)^2 + \lambda \sum_{i=1}^{d} |w_i|$$

$$= \sum_{i=1}^{n} \left((x_i^T \cdot w)^2 - 2y_i \cdot x_i^T \cdot w + y_i^2 \right) + \lambda \sum_{i=1}^{d} |w_i|$$

$$= \|y\|_2^2 + \sum_{i=1}^{d} \left(\|w_i X_{*i}\|_2^2 - 2w_i X_{*i}^T \cdot y + \lambda |w_i| \right)$$

$$= \|y\|_2^2 + \sum_{i=1}^{d} \left(n w_i^2 - 2w_i \cdot X_{*i}^T \cdot y + \lambda |w_i| \right)$$

(2) $\nabla_w J(w^*) = [2nw_1^* - 2X_{*1}^T y + \lambda, \dots, 2n \cdot w_d^* - 2X_{*d}^T \cdot y + \lambda] = 0$

so $2nw_i^* - 2X_{*i}^T \cdot y + \lambda = 0$

$$w_i^* = \frac{2X_{*i}^T \cdot y - \lambda}{2n} \quad (>0)$$

(3) $\nabla_w J(w^*) = [2nw_1^* - 2X_{*1}^T \cdot y - \lambda, \dots, 2nw_d^* - 2X_{*d}^T \cdot y - \lambda] = 0$

so $2nw_i^* - 2X_{*i}^T \cdot y - \lambda = 0$

$$w_i^* = \frac{2X_{*i}^T \cdot y + \lambda}{2n} \quad (<0)$$

(4) Using contrappositive, if $2X_{*i}^T \cdot y - \lambda \leq 0$, then w_i^* is not positive. Also, if $2X_{*i}^T \cdot y + \lambda \geq 0$, then w_i^* is not negative. If we satisfy both statement, w_i^* must be zero. Thus, the condition is:

$$-\lambda \leq 2X_{*i}^T \cdot y \leq \lambda$$

(5) $J(w) = |Xw - y|^2 + \lambda \|w\|_2^2$

$$= \|y\|_2^2 + \sum_{i=1}^{d} \left(n w_i^2 - 2w_i + X_{*i}^T \cdot y + \lambda w_i^2 \right)$$

$\nabla J(w^*) = 0$

$2nw_i^* - 2X_{*i}^T \cdot y + 2\lambda w_i^* = 0$ for any w_i^*

$$w_i^* = \frac{X_{*i}^T \cdot y}{n + \lambda} \quad (n + \lambda \neq 0 \text{ as } n > 0, \lambda > 0)$$

$\therefore$ If $X_{*i}^T \cdot y = 0$, then w_i^*

From part (4), it was range of $X_{*i}^T \cdot y$, but in part (5), $X_{*i}^T \cdot y$ is fixed to be zero.

$\boxed{Q3}$ (a) $\nabla|w|^4 = \nabla\left(\sum_{j=1}^{d} w_j^2\right)^2 = \begin{bmatrix} \frac{\partial}{\partial w_1}\left(\sum_{j=1}^{d} w_j^2\right)^2 \\ \vdots \\ \frac{\partial}{\partial w_d}\left(\sum_{j=1}^{d} w_j^2\right)^2 \end{bmatrix} = \begin{bmatrix} 2w_1\left(\sum_{j=1}^{d} w_j^2\right)\cdot 2 \\ \vdots \\ 2w_d\left(\sum_{j=1}^{d} w_j^2\right)\cdot 2 \end{bmatrix} = 4|w|^2 \cdot w$

$\nabla_w |Xw-y|^4 = 4|Xw-y|^2 \cdot X^T(Xw-y)$

(b) $\nabla_w^2 \left(|Xw-y|^4 + \lambda|w|^2\right) = \nabla_w \, 4|Xw-y|_2^2 (X^TXw - X^Ty) + 2\lambda I_{d\times d}$

$= 4\left((2X^TXw-2X^Ty)(X^TXw-X^Ty)^T + |Xw-y|_2^2 X^TX\right) + 2\lambda I_{d\times d}$

$= 4\left(3(X^TXw-X^Ty)(X^TXw-X^Ty)^T\right) + 2\lambda I_{d\times d}$

$= 12(X^TXw-X^Ty)(X^TXw-X^Ty)^T = 12\left|X^TXw-X^Ty\right| + 2\lambda I_{d\times d} > 0$

$\therefore$ The cost function is convex, so there should be a unique w^* that minimizes the cost function

$\nabla_w\left(|Xw-y|^4 + \lambda|w|^2\right) = 4|Xw^*-y|^2 X^T(Xw^*-y) + 2\lambda w^* = 0.$

$\therefore w^* = -\frac{1}{2\lambda} \cdot 4|Xw^*-y|^2 X^T(Xw^*-y)$

$= \sum_{i=1}^{n} -\frac{4}{\lambda}|Xw^*-y|^2 X_i \cdot (X_i \cdot w^* - y_i)$

$= \sum_{i=1}^{n} -\frac{4}{\lambda}|Xw^*-y|^2 (X_i \cdot w^* - y_i) \cdot X_i$

Let $a_i = -\frac{4}{\lambda}|Xw^*-y|^2 (X_i \cdot w^* - y_i)$

Then $w^* = \sum_{i=1}^{n} a_i X_i$

(c) Since the cost function is convex, the optimal solution happens when the gradient is zero. Since L is of w^TX_i and we are differentiating it with w, so there will always be X_i term by chain rule. The gradient must have $d\times 1$ dimension, which is same with X_i. Thus, all the terms of gradient except X_i must be scalar. There fore, the optimal solution has the form $w^* = \sum_{i=1}^{n} a_i X_i$.

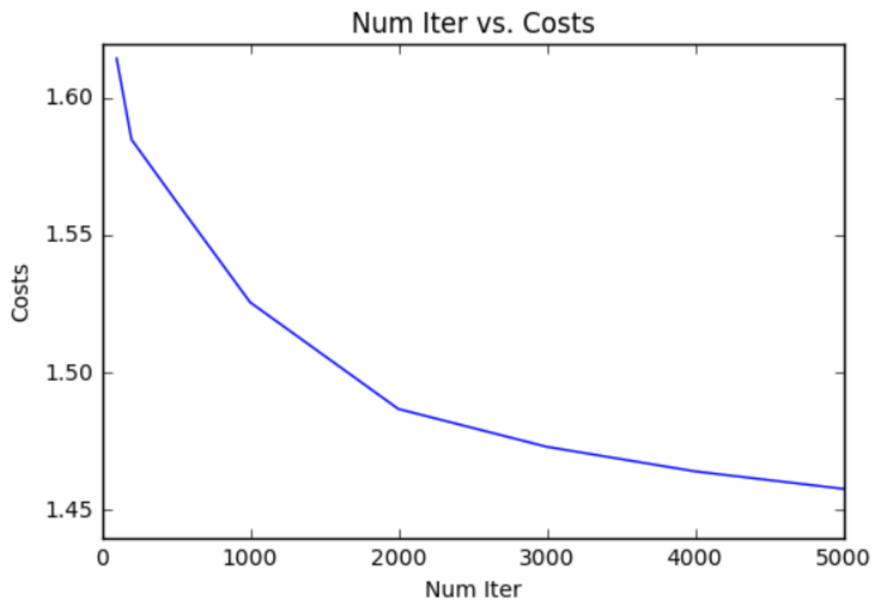
If the cost function is concave, then w that makes the gradient zero is maximizer of the cost function. Thus, it is not guaranteed that the optimal solution has the form $w^* = \sum_{i=1}^{n} a_i X_i$

Q7 The problem is that the number of spam counted is reset on midnight, so in the graph, the number of spam rapidly increases at the both end points. Thus, if we change the time of resetting to noon from midnight, the graph will increases only around the mid point (midnight). We can classify this new feature with quadratic kernel and the result is expected to be improved.

Q4 part1:

Derivation is same with question 1 part 1.

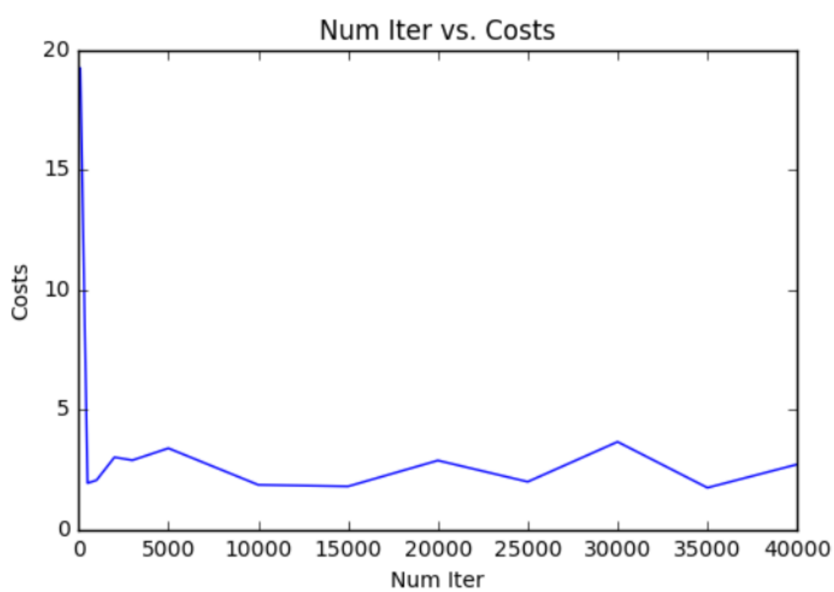
w = w – eps*sum(-1*(yi-s(Xi^T*w)*Xi)+2*lambda*w where s is logistic function



Q4 part2:

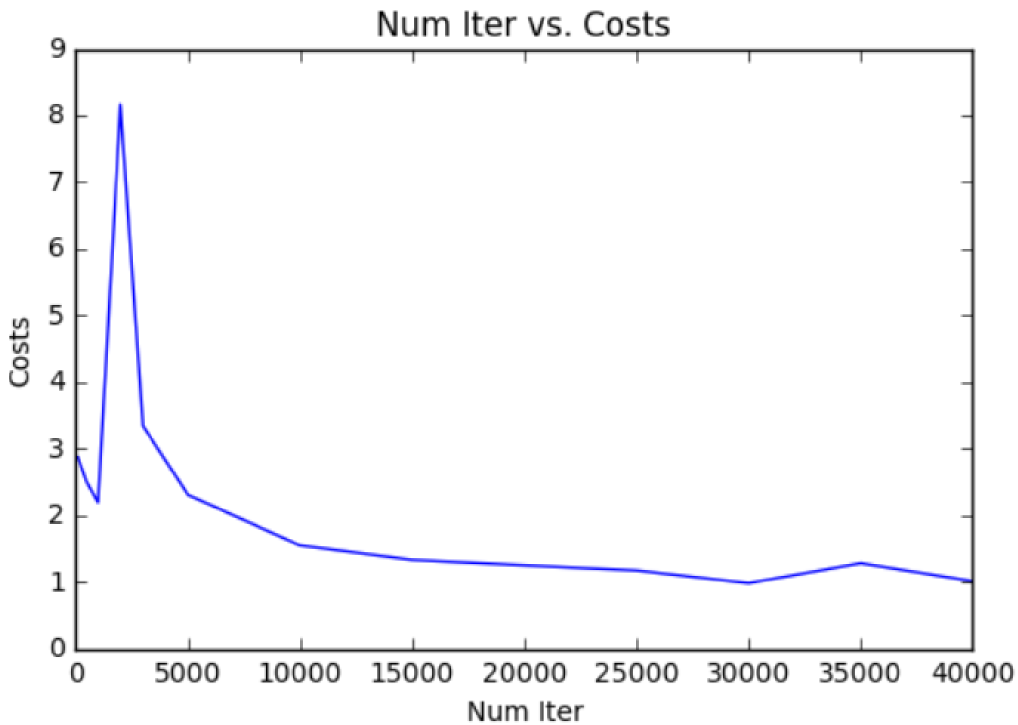
Derivation is same with question 1 part 1 except summation over n.

w = w + (yi-s(Xi^T*w))*Xi+2*lambda*w where i is randomly selected and s is logistic function.



Q4 part3:

The new strategy works better than having a constant epsilon value.



Q4 part4:

Kaggle ID: YONG-CHAN_SHIN

Score: 0.93952 (PASSED)

I tried with global lists of several lambda and epsilon values and find out which combination of these values work best (as you can see in the code). After finding the best value, use the local lists (list inside the functions) of the best value so that we do not have to iterate over the experimented values again.

Code:

```
import scipy.io
import scipy.special
import numpy as np
import math
from random import randint
import matplotlib.pyplot as plt
import pandas as pd

def safe_log(x):
    return np.log(x.clip(min=0.00000000001))

def cost_func(y,X,w,lamb):
    tot=0
    z = []
    samples = np.dot(X,w)
    num_samples = int(samples.shape[0])
    #print "shape of dotXw",np.dot(X,w).shape
    for Xiw in samples:
        z.append(logistic(Xiw))
    for i in range(num_samples):
        add = y[i]*safe_log(z[i])+(1-y[i])*safe_log(1-z[i])
        tot+=add
    return (-1)*tot/float(num_samples)+lamb*(np.linalg.norm(w)**2)

def compute_score(labels,pred):
        error_count = 0
        for tup in zip(pred,labels):
                if tup[0]!=tup[1]:
                        error_count+=1
        return float(labels.shape[0]-error_count)/float(labels.shape[0])
```

```
def logistic(s):
	#print "s:",s
	return scipy.special.expit(s)
#	try:
#		return 1/float(1+math.exp((-1)*s))
#	except OverflowError:
#		return 1
#	except UnderflowError:
#		return 0

def gradient(lam,w,Xi,yi):
#	print "w shape:",w.shape
#	print "Xi shape:",Xi.shape
#	print "yi shape:",yi.shape
	return (-1)*(yi-logistic(np.dot(Xi,w)))*Xi+2*lam*w

def batch(w0,num_iter,training_set,eps_list,lambda_list):
	print "current num_iter",num_iter
	X = training_set[:,:training_set.shape[1]-1]
	y = training_set[:,training_set.shape[1]-1]
	w_list = []
	counter=0
	for eps in eps_list:
		for lam in lambda_list:
			w = w0
			for j in range(num_iter):
				n_R = np.array([eps*gradient(lam,w,X[i],y[i]) for i in range(n)])
				#print "n_R shape:",n_R.shape
				#print "avg shape:",np.mean(n_R,axis=0).shape
				w = w - np.mean(n_R,axis=0)
				#print "in numiter iter:",j
			w_list.append(w)
```

```
            counter+=1
            #print "In batch, in nested loop, counter:",counter
    return w_list

def batch_part4(w0,num_iter,training_set,eps_list,lambda_list):
    print "current num_iter",num_iter
    X = training_set[:,:training_set.shape[1]-1]
    y = training_set[:,training_set.shape[1]-1]
    w_list = []
    counter=0
    for eps in eps_list:
        for lam in lambda_list:
            w = w0
            for j in range(num_iter):
                if j!=0:
                    eps = 1/float(j)
                n_R = np.array([eps*gradient(lam,w,X[i],y[i]) for i in range(n)])
                #print "n_R shape:",n_R.shape
                #print "avg shape:",np.mean(n_R,axis=0).shape
                w = w - np.mean(n_R,axis=0)
                #print "in numiter iter:",j
            w_list.append(w)
            counter+=1
            #print "In batch, in nested loop, counter:",counter
    return w_list

def stoch(w0,num_iter,training_set,eps_list,lambda_list):
    X = training_set[:,:training_set.shape[1]-1]
    y = training_set[:,training_set.shape[1]-1]
    w_list = []
    for eps in eps_list:
        for lam in lambda_list:
```

```
            w = w0
            for _ in range(num_iter):
                i = randint(0,X.shape[0]-1)
                w = w - eps*gradient(lam,w,X[i],y[i])
            w_list.append(w)
    return w_list

def stoch_part3(w0,num_iter,training_set,eps_list,lambda_list):
    X = training_set[:,:training_set.shape[1]-1]
    y = training_set[:,training_set.shape[1]-1]
    w_list = []
    for eps in eps_list:
        for lam in lambda_list:
            w = w0
            for t in range(num_iter):
                if t!=0:
                    eps = 1/np.sqrt(t)
                i = randint(0,X.shape[0]-1)
                w = w - eps*gradient(lam,w,X[i],y[i])
            w_list.append(w)
    return w_list

mat_contents = scipy.io.loadmat('hw4_wine_dist/data.mat')
test_set = mat_contents['X_test']
feature_names = mat_contents['description']
training_set = mat_contents['X']
#training_set = np.array([sample/np.linalg.norm(sample) for sample in training_set])
training_label = mat_contents['y']
training_data = np.concatenate((training_set,training_label),axis=1)
np.random.shuffle(training_data)
validation_set = training_data[:training_data.shape[0]//5,:]
training_set = training_data[training_data.shape[0]//5:,:]
```

```
nums_iter_batch = [100,200,500,1000,2000,3000,4000,5000]
#nums_iter_batch = [10,20,30,50,100]
#nums_iter_batch = [1000]
nums_iter_stoch = [100,500,1000,2000,3000,5000,10000,15000,20000,25000,30000,35000,40000]
eps_list = [0.001, 0.01, 0.1, 1.0]
lambda_list = [0.001, 0.01, 0.1, 1.0]
n = training_set.shape[0]
d = training_set.shape[1]-1
w0 = np.array([0.0 for i in range(d)])


def q1():
    eps_list = [0.01] # found by test
    lambda_list = [0.01] # found by test
    errors = []
    costs = []
    for num_iter in nums_iter_batch:
        errors_w_wise = []
        #w_list is len(eps_list) by len(lambda_list)
        w_list = batch(w0,num_iter,training_set,eps_list,lambda_list)
        X = validation_set[:,:validation_set.shape[1]-1]
        y = validation_set[:,validation_set.shape[1]-1]
        for w in w_list:
            pred = []
            for i in range(X.shape[0]):
                if logistic(np.dot(X[i],w)) < 0.5:
                    pred.append(0)
                else:
                    pred.append(1)
            score = compute_score(y,pred)
            errors_w_wise.append(1.0-score)
        arg_max = np.argmax(errors_w_wise)
        w_max = w_list[arg_max]
```

```
        eps_index = arg_max//len(eps_list)-1
        lamb_index = arg_max%len(lambda_list)-1
        errors.append(errors_w_wise[arg_max])
        print "Best eps val:",eps_list[eps_index]
        print "Best lambda val:",lambda_list[lamb_index]
        costs.append(cost_func(y,X,w_max,lambda_list[lamb_index]))

    print "Errors:",errors
    print "Costs:",costs
    plt.plot(nums_iter_batch,costs)
    plt.title('Num Iter vs. Costs')
    plt.xlabel('Num Iter')
    plt.ylabel('Costs')
    plt.show()
    plt.plot(nums_iter_batch,errors)
    plt.title('Num Iter vs. Error Rate')
    plt.xlabel('Num Iter')
    plt.ylabel('Error Rate')
    plt.show()

def q2():
    eps_list = [0.01] # found by test
    lambda_list = [0.01] # found by test
    errors = []
    costs = []
    for num_iter in nums_iter_stoch:
        errors_w_wise = []
        #w_list is len(eps_list) by len(lambda_list)
        w_list = stoch(w0,num_iter,training_set,eps_list,lambda_list)
        X = validation_set[:,:validation_set.shape[1]-1]
        y = validation_set[:,validation_set.shape[1]-1]
        for w in w_list:
```

```
            pred = []
            for i in range(X.shape[0]):
                if logistic(np.dot(X[i],w)) < 0.5:
                    pred.append(0)
                else:
                    pred.append(1)
            score = compute_score(y,pred)
            errors_w_wise.append(1.0-score)
        arg_max = np.argmax(errors_w_wise)
        w_max = w_list[arg_max]
        eps_index = arg_max//len(eps_list)-1
        lamb_index = arg_max%len(lambda_list)-1
        errors.append(errors_w_wise[arg_max])
        print "Best eps val:",eps_list[eps_index]
        print "Best lambda val:",lambda_list[lamb_index]
        costs.append(cost_func(y,X,w_max,lambda_list[lamb_index]))

    print "Errors:",errors
    print "Costs:",costs
    plt.plot(nums_iter_stoch,costs)
    plt.title('Num Iter vs. Costs')
    plt.xlabel('Num Iter')
    plt.ylabel('Costs')
    plt.show()
    plt.plot(nums_iter_stoch,errors)
    plt.title('Num Iter vs. Errors')
    plt.xlabel('Num Iter')
    plt.ylabel('Errors')
    plt.show()

def q3():
    eps_list = [0.01] # found by test
```

```
lambda_list = [0.001] # found by test
errors = []
costs = []
for num_iter in nums_iter_stoch:
    errors_w_wise = []
    #w_list is len(eps_list) by len(lambda_list)
    w_list = stoch_part3(w0,num_iter,training_set,eps_list,lambda_list)
    X = validation_set[:,:validation_set.shape[1]-1]
    y = validation_set[:,validation_set.shape[1]-1]
    for w in w_list:
        pred = []
        for i in range(X.shape[0]):
            if logistic(np.dot(X[i],w)) < 0.5:
                pred.append(0)
            else:
                pred.append(1)
        score = compute_score(y,pred)
        errors_w_wise.append(1.0-score)
    arg_max = np.argmax(errors_w_wise)
    w_max = w_list[arg_max]
    eps_index = arg_max//len(eps_list)-1
    lamb_index = arg_max%len(lambda_list)-1
    errors.append(errors_w_wise[arg_max])
    print "Best eps val:",eps_list[eps_index]
    print "Best lambda val:",lambda_list[lamb_index]
    costs.append(cost_func(y,X,w_max,lambda_list[lamb_index]))

print "Errors:",errors
print "Costs:",costs
plt.plot(nums_iter_stoch,costs)
plt.title('Num Iter vs. Costs')
plt.xlabel('Num Iter')
```

```
    plt.ylabel('Costs')
    plt.show()
    plt.plot(nums_iter_stoch,errors)
    plt.title('Num Iter vs. Errors')
    plt.xlabel('Num Iter')
    plt.ylabel('Errors')
    plt.show()

def q4():
    eps_list = [0.01] # found by test
    lambda_list = [0.001] # found by test
    nums_iter_stoch = [40000]
    for num_iter in nums_iter_stoch:
        #w_list is len(eps_list) by len(lambda_list)
        w_list = stoch_part3(w0,num_iter,training_data,eps_list,lambda_list)
        X = test_set
        for w in w_list:
            pred = []
            for i in range(X.shape[0]):
                if logistic(np.dot(X[i],w)) < 0.5:
                    pred.append(0)
                else:
                    pred.append(1)
            table = {"Category":pred, "Id":np.arange(0,len(pred))}
            output = pd.DataFrame(data=table)
            output.to_csv("kaggle_ycls_hw4.csv",index=False)
            print "file created"

def prob1():
    X = np.array([[0,3,1],[1,3,1],[0,1,1],[1,1,1]])
    y = np.transpose(np.array([1,1,0,0]))
    w0 = np.transpose(np.array([-2,1,0]))
```

```
    n = X.shape[0]
    d = X.shape[1]
    lamb = 0.07
    S = lambda wi: np.transpose(np.array([logistic(np.dot(X[i],wi)) for i in range(n)]))
    S0 = S(w0)
    hessian = lambda Si: 2*lamb*np.identity(d)+np.dot(Si,np.transpose(np.array([1 for _ in range(n)]))-Si)*np.dot(np.transpose(X),X)
    gradient = lambda Si: 2*lamb*w-np.dot(np.transpose(X),y-Si)
    update = lambda Si: np.dot(np.linalg.inv(hessian(Si)),gradient(Si))
    w1 = w0-update(S0)
    S1 = S(w1)
    w2 = w1-update(S1)
    print "S0:",S0
    print "w1:",w1
    print "S1:",S1
    print "w2:",w2

#prob1()
#q1()
#q2()
q3()
#q4()
```