EECS 545 Homework 3 Yuan Yin

2) Robuct Regression

(a) According to Majorize - Minimize Algorithm, we find a majorizing function $\bar{J}(\vec{\theta}, \vec{\theta}_t)$ and then iterate $\vec{\theta}_t$ with solution for $\vec{\theta}_{t+1} = \arg\min \bar{J}(\vec{\theta}, \vec{\theta}_t)$

According to the lemma in notes, we have such $\bar{J}$ s.t.

$$\bar{J}(\vec{\theta}, \vec{\theta}_t) = \sum_{i=1}^{n} \bar{\rho}(y_i - \vec{w}^T\vec{x}_i - b)$$

Where $\bar{\rho}(r) = \rho(r_{t,i}) - \frac{1}{2} r_{t,i}\psi(r_{t,i}) + \frac{1}{2}\frac{\psi(r_{t,i})}{r_{t,i}} r^2$

$$\psi(r) = \rho'(r), \quad r_{t,i} = y_i - \vec{w}_t^T \vec{x}_i - b_t$$

It's nothing but

$$\vec{\theta}_{t+1} = \arg\min \left(\sum_{i=1}^{n} \left(\rho(r_{t,i}) - \frac{1}{2} r_{t,i}\psi(r_{t,i}) + \frac{1}{2}\frac{\psi(r_{t,i})}{r_{t,i}} r^2 \right)\right)$$

since for time t, $r_{t,i}$ is known (can be computed)

Then $\vec{\theta}_{t+1} = \arg\min \sum_{i=1}^{n} \frac{1}{2}\frac{\psi(r_{t,i})}{r_{t,i}} r^2$

Suppose $c_{t,i} = \frac{1}{2}\frac{\psi(r_{t,i})}{r_{t,i}}$, then

$$\vec{\theta}_{t+1} = \arg\min_{\vec{\theta}} \sum_{i=1}^{n} c_{t,i} (y_i - \vec{\theta}^T\vec{x}_i)^2$$

Since that $c_{t,i}$ is changing with iteration time t changes, we can think MM algorithm as "iteratively reweighted least squares"

Explain of robustness: From the lemma we know that $c_{t,i} = \frac{1}{2}\frac{\psi(r_{t,i})}{r_{t,i}}$ is nonincreasing for $r_{t,i} > 0$. That is to say, for outliers who have larger error $r_{t,i}$, the weight $c_{t,i}$ will be smaller, (Noting that since $\rho(r)$ is symmetric from the lemma, weights $c_{t,i}$ is also symmetric, so this means actually when error $r_{t,i}$ gets further against zero, weights get smaller), but for least square regression, all weights are equal so there is no difference between outliers and inliers. Thus this algorithm achieves robustness since it decreases the importance for outliers.

3) Logistic Regression as ERM

Recall logistic regression: (for labels $Y \in \{-1, 1\}$)

$$f^*(\vec{x}) = \begin{cases} 1 & \text{if } \eta(x) \geq \frac{1}{2} \\ -1 & \text{otherwise.} \end{cases}$$

$\eta(\vec{x}) := Pr\{Y=1 \mid X=\vec{x}\}$ and we know $\eta(\vec{x}) = \dfrac{1}{1+\exp[-(w^T x+b)]}$

$$\Rightarrow Pr\{Y=-1 \mid X=\vec{x}\} = 1-\eta(\vec{x}) = \frac{1}{1+\exp(w^T x+b)}$$

$$\Rightarrow P(y \mid \vec{x}; \vec{\theta}) = \begin{cases} 1-\eta(\vec{x}) & \text{if } y=-1 \\ \eta(\vec{x}) & \text{if } y=1 \end{cases}$$

$$= \frac{1}{1+\exp[-y(w^T x+b)]}$$

Then the log-likelihood of $\vec{\theta}$ is defined to be.

$$l(\vec{\theta}) = \log L(\vec{\theta}) \underset{\Uparrow}{=} -\sum_{i=1}^{n} \log(1+\exp[-y_i(w^T x_i+b)])$$

$$\left[= \log\left(\prod_{i=1}^{n} P(y_i \mid \vec{x}_i; \vec{\theta})\right)\right]$$

As for ERM with the logistic loss is $\frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y(w^T x_i+b)))$, which is just proportional to the negative log likelihood for logistic regression

2.

4) Subgradient methods for the optimal soft margin hyperplane

since $J(\vec{w}, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \vec{w}^T \vec{x}_i + b) + \frac{\lambda}{2} \|\vec{w}\|^2$

Then for $J_i(\vec{w}, b)$ satisfying $J(\vec{w}, b) = \sum_{i=1}^{n} J_i(\vec{w}, b)$

$\Rightarrow \underline{J_i(\vec{w}, b) = \frac{1}{n} L(y_i, \vec{w}^T \vec{x}_i + b) + \frac{\lambda}{2n} \|\vec{w}\|^2}$ where $L(y, t) = \max\{0, 1-yt\}$

Now determine $\vec{u}_i$: $\vec{u}_i = \nabla J_i(\vec{w}, b) = \nabla J_i(\vec{\theta})$

suppose $g(z) = \max\{0, z\}$ $h_i(\vec{z}) = 1 - y_i(\vec{z} \cdot \tilde{x}_i)$

where $\tilde{x}_i = [1 \ \ \vec{x}_i]^T$

Then $\vec{u}_i = \nabla J_i(\vec{\theta})$

$= \frac{1}{n} \nabla[g(h_i(\vec{\theta}))] + \frac{\lambda}{n}[0 \ \ \vec{w}^T]^T$

$= \frac{1}{n}[\nabla h_i(\vec{\theta}) \cdot g'(h_i(\vec{\theta}))] + \frac{\lambda}{n}[0, \vec{w}^T]^T$

$= \frac{1}{n}(-y_i \cdot \tilde{x}_i) \cdot \text{scale} + \frac{\lambda}{n}[0 \ \ \vec{w}^T]^T$

where $\text{Scale} = \begin{cases} 1 & \text{if } 1 - y_i(\vec{\theta}^T \hat{x}_i) > 0 \\ 0 & \text{if } 1 - y_i(\vec{\theta}^T \hat{x}_i) < 0 \\ \text{random number in } (0,1) & \text{if } 1 - y_i(\vec{\theta}^T \hat{x}_i) = 0 \end{cases}$

$\hat{x}_i = [1 \ \ \vec{x}_i]^T$

5) ERM Losses

(a) merge the condition:

$\left.\begin{array}{l} y_i(\vec{w}^T \vec{x}_i + b) \geq 1 - \xi_i \quad \forall i \\ \xi_i \geq 0 \quad \forall i \end{array}\right\} \Leftrightarrow \xi_i \geq \max\{0, 1 - y_i(\vec{w}^T \vec{x}_i + b)\}$

Thus obj. fun deduces to:

$$\min_{\vec{w}, b, \xi} \frac{1}{2}\|\vec{w}\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i^2$$

$$\text{s.t. } \xi_i \geq \max\{0, 1 - y_i(\vec{w}^T \vec{x}_i + b)\}$$

3.

$$\Rightarrow \min_{\vec{w}, b, \vec{\xi}} \frac{1}{2}\|\vec{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i^2$$

$$\text{s.t. } \xi_i = \max\{0, 1 - y_i(\vec{w}^T\vec{x}_i + b)\}.$$

It associates with ERM with squared ~~hindge~~ hinge loss

(b). if $\xi_i \geq 0$ dropped,

obj.fun reduces to: $\min_{\vec{w}, b, \vec{\xi}} \frac{1}{2}\|\vec{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i^2$

$\text{s.t. } \xi_i \geq 1 - y_i(\vec{w}^T\vec{x}_i + b).$

suppose if $1 - y_i(\vec{w}^T\vec{x}_i + b) < 0$. so that ξ_i can be ~~choosen~~ chosen as less than zero satisfying the constraints above. $(\xi_i \geq 1 - y_i(\vec{w}\vec{x}_i + b))$

Then it's obvious that for $\xi_i < 0$. $\xi_i^2 > 0^2$.

for we choose these ξ_i to be zero still satisfies constraints above but with smaller objective value

Thus there is no need to specify $\xi_i \geq 0$ because we will choose $\xi_i = 0$ for those ξ_i can be less than zero.

(C) advantage: Since this loss uses squared slack variables, to minimize the objective function, there will be more penlty to data points which violate the margin.

disvantage: But also more penlty means there will be less robustness than hinge loss.

EECS 545 Homework 3

Yuan Yin

October 18, 2018

Problem 1

The result is:

```
Estimated omega is: 0.6486572567576243 -0.06324140302684156
Estimated b is: -34.083439600883615
Using remainder for estimating MSE is: 21.485893070546425
Predicted response at the input x=[100 100] is: 24.458145772194662
```

Problem 2

(b) & (c)

The code and result is as follows:

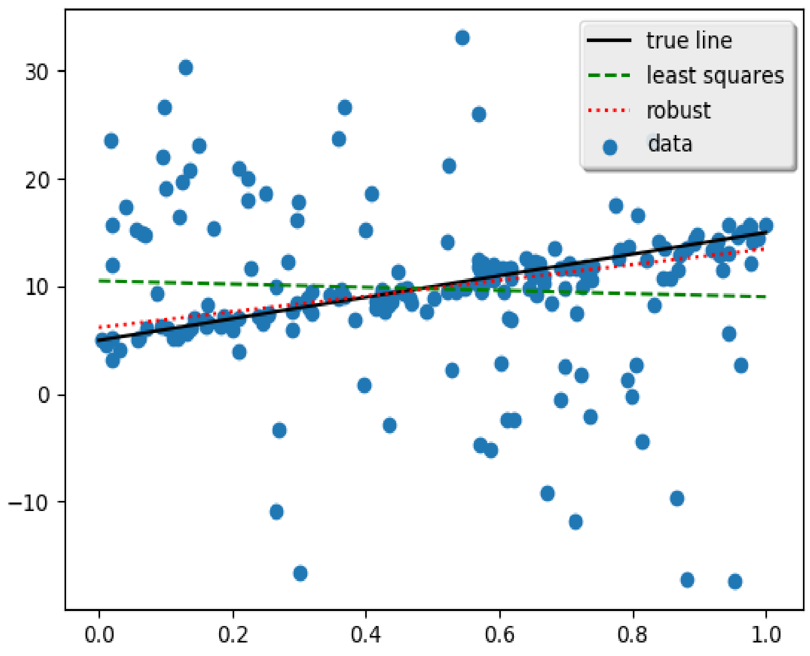
```
import numpy as np
import matplotlib.pyplot as plt
import pylab as pl

n = 200
np.random.seed(0) # Seed the random number generator
x = np.random.rand(n,1)
z = np.zeros([n,1])
k = n * 0.4
rp = np.random.permutation(n)
outlier_subset = rp[1:int(k)]
z[outlier_subset] = 1 # outliers
y = (1 - z) * (10 * x + 5 + np.random.randn(n,1)) + z * (20 - 20 * x + 10 *
    np.random.randn(n,1))

# Plot data and true line
plt.scatter(x, y, label = 'data')
```

```
t = pl.frange(0,1,0.01)
plt.plot(t, 10*t+5, 'k-', label = 'true line')

# Add your code for ordinary least squares below
x_ols = np.mat(x); y_ols = np.mat(y) # They are all 200*1 matrix
y_mdf = y_ols - np.mean(y_ols); x_mdf = x_ols - np.mean(x_ols)
A = 2 * (x_mdf.T * x_mdf); r = -2 * x_mdf.T * y_mdf; c = y_mdf.T * y_mdf
w_ols = -(A.I * r)[0,0]; b_ols = np.mean(y_ols) - w_ols * np.mean(x_ols)
print("Parameters of OLS are: w: ",w_ols, "b: ", b_ols)

plt.plot(t, w_ols*t+b_ols, 'g--', label = 'least squares')

#######################################################
# helper function to solve weighted least squares
    # add your code here
def wls(x,y,c):
    add = 1
    x_add = np.insert(x, 0, values=add, axis=1)
    theta = (x_add.T * c * x_add).I * x_add.T * c * y
    b = theta[0]; w = theta[1]
    return w, b

# Add your code for robust regression MM algorithm below
x_rob = np.mat(x); y_rob = np.mat(y)
w_rob = 0; b_rob = 0
error = 1
while error > 0.01:
    c = np.eye(len(x_rob))
    for i in range(len(x_rob)):
        r_t_i = y_rob[i] - w_rob * x_rob[i] - b_rob
        c[i,i] = 1/(2*np.sqrt(1+r_t_i**2))
    w_new, b_new = wls(x_rob, y_rob, c)
    error = np.sqrt((w_new - w_rob)**2 + (b_new - b_rob)**2)
    w_rob, b_rob = w_new[0,0], b_new[0,0]

print("Parameters of ROB are: w: ",w_rob, "b: ", b_rob)
plt.plot(t, w_rob*t+b_rob, 'r:', label = 'robust')
legend = plt.legend(loc='upper right', shadow=True)
plt.show()
```

The result is:

```
Parameters of OLS are: w: -1.4749427077811519 b: 10.519708850514508
Parameters of robust regression are: w: 7.302235544182917 b: 6.186315755157274
```

The plot which shows the data, the true line, the OLS estimate and the robust estimate is as bellow:



Problem 4

(b)

The code and the result is as follows:

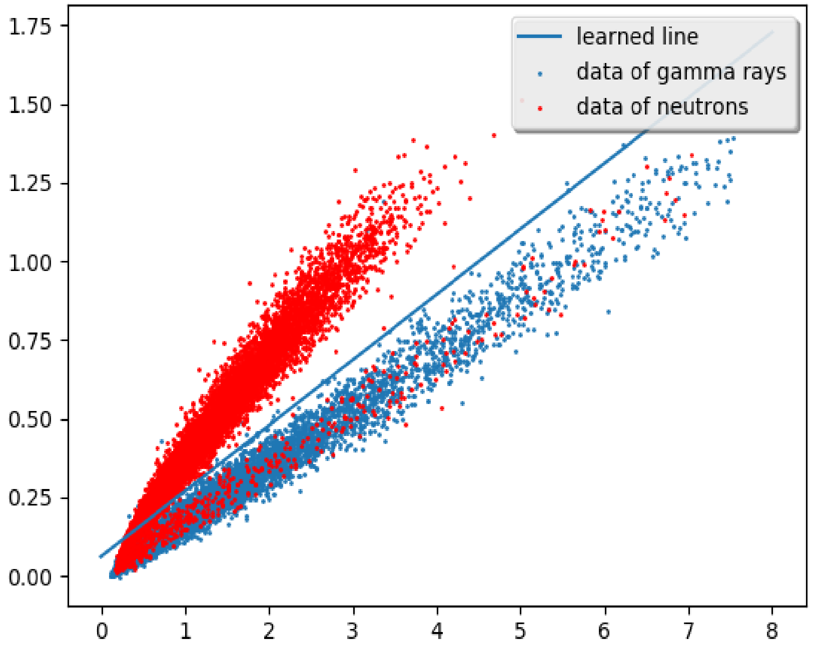
```
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import pylab as pl

np.random.seed(0) # Seed the random number generator
# Load the data
nuclear = sio.loadmat('nuclear.mat')
x = nuclear['x']
y = nuclear['y'] # 1 represents neutrons and -1 represents gamma rays
d,n = x.shape

# Plot data
gamma_rays = []; neutrons = []
```

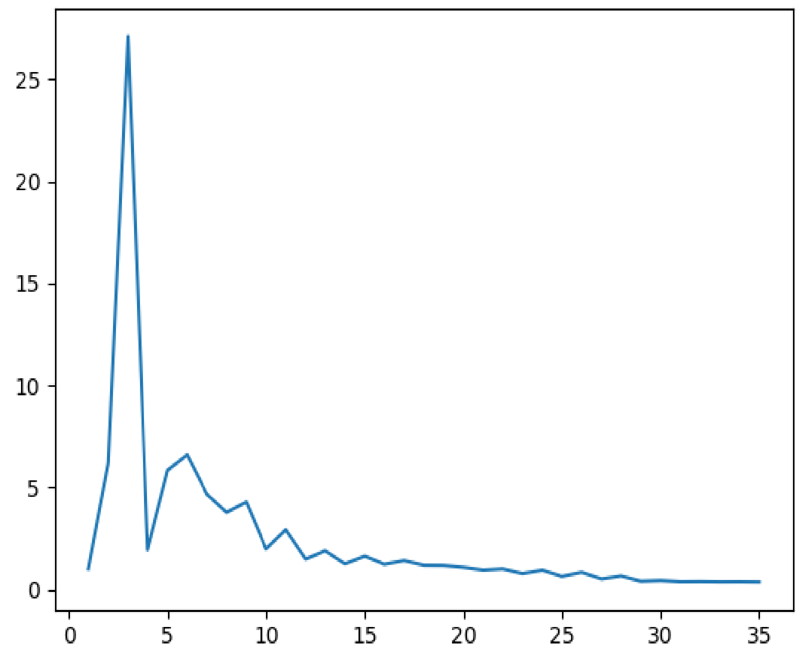
```python
for i in range(n):
    if y[0,i] == 1:
        neutrons.append(i)
    else:
        gamma_rays.append(i)
plt.figure()
plt.scatter(x[0,gamma_rays], x[1,gamma_rays], s = 1, label = 'data of gamma rays')
plt.scatter(x[0,neutrons], x[1,neutrons], s = 1, c = 'r', label = 'data of neutrons')

# Initialization and plot learned line
x = np.mat(x); y = np.mat(y)
add = 1; J = []
x_tilda = np.insert(x, 0, values=add, axis=0)
lamda = 0.001; omega = np.zeros(d); b = 0
Iteration = 35
for j in range(Iteration):
    u = 0; alpha = 100/(j+1); Loss = 0
    for i in range(n):
        h = 1 - y[0,i] * (np.dot(omega, x[:,i]) + b)
        if h[0,0] > 0:
            scale = 1
        elif h[0,0] < 0:
            scale = 0
        else:
            scale = np.random.rand()
        u += 1/n * (-y[0,i] * x_tilda[:,i]) * scale + lamda / n * np.mat([0, omega[0],
            omega[1]]).T
        Loss += max(0, h[0,0])
    J.append(Loss/n + lamda/2 * np.dot(omega, omega))
    b -= alpha * u[0]; omega[0] -= alpha * u[1]; omega[1] -= alpha * u[2]

print("Parameters of estimated hyperplane are: b = ", b[0,0], "w_1 = ", omega[0], "w_2 = ",
    omega[1])
print("the minimum achieved value of the objective function is: ", min(J))
t = pl.frange(0,8,0.01)
n = pl.frange(1,Iteration)
plt.plot(t, -omega[0]/omega[1] * t - b[0,0]/omega[1], label = 'learned line')
legend = plt.legend(loc='upper right', shadow=True)
# Plot the objective function
plt.figure()
plt.plot(n, J, label = 'Objective Function')
plt.show()
```

The result is:

```
Parameters of estimated hyperplane are:
b = -1.1543275085302598 w_1 = -3.8809615376043824 w_2 = 18.631005972691405
the minimum achieved value of the objective function is: 0.3590574177462417
```

The plots showing the data and the learned line, and showing J as a function of iteration number is as bellow:





(c)

the code and the result is as follows:

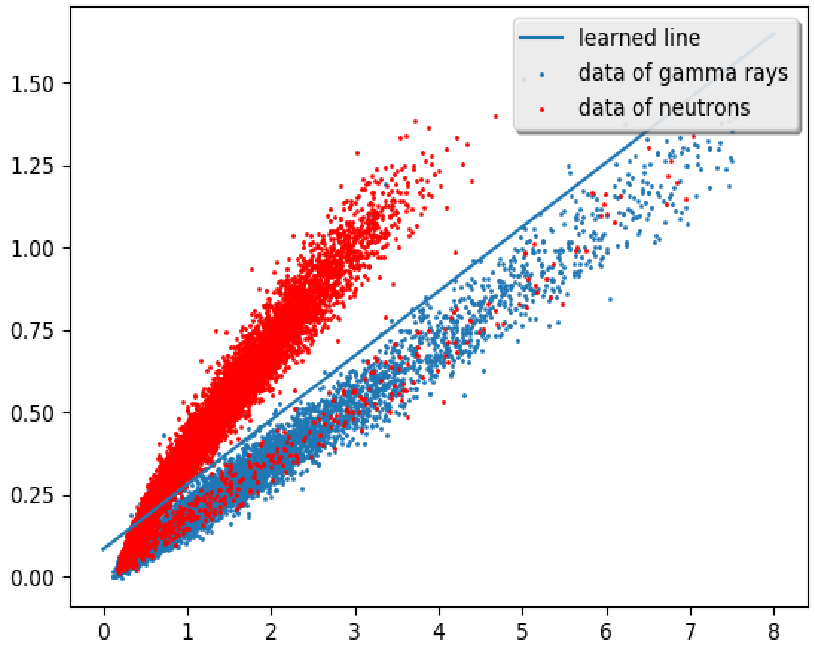
```
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import pylab as pl

np.random.seed(0) # Seed the random number generator
# Load the data
nuclear = sio.loadmat('nuclear.mat')
x = nuclear['x']
y = nuclear['y'] # 1 represents neutrons and -1 represents gamma rays
d,n = x.shape

# Plot data
gamma_rays = []; neutrons = []
for i in range(n):
    if y[0,i] == 1:
        neutrons.append(i)
    else:
        gamma_rays.append(i)
```

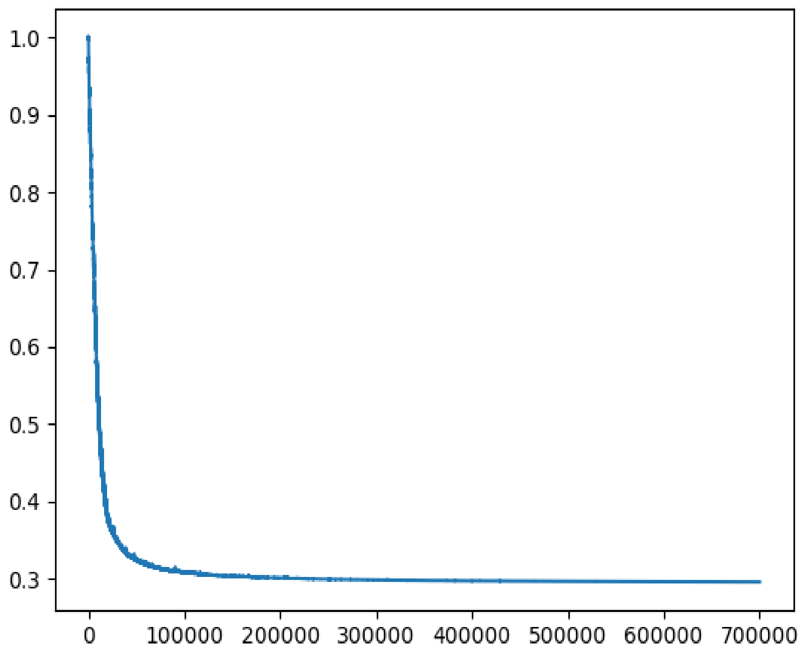
```
plt.figure()
plt.scatter(x[0,gamma_rays], x[1,gamma_rays], s = 1, label = 'data of gamma rays')
plt.scatter(x[0,neutrons], x[1,neutrons], s = 1, c = 'r', label = 'data of neutrons')

# Initialization and plot learned line with stochastic sub-gradient method
x = np.mat(x); y = np.mat(y)
add = 1; J = []
x_tilda = np.insert(x, 0, values=add, axis=0)
lamda = 0.001; omega = np.zeros(d); b = 0
Iteration = 35
for j in range(Iteration):
    ui = 0; alpha = 100/(j+1); N = np.random.permutation(n); Loss = 0
    for i in N:
        h = 1 - y[0,i] * (np.dot(omega, x[:,i]) + b)
        if h[0,0] > 0:
            scale = 1
        elif h[0,0] < 0:
            scale = 0
        else:
            scale = np.random.rand()
        ui = 1/n * (-y[0,i] * x_tilda[:,i]) * scale + lamda / n * np.mat([0, omega[0],
            omega[1]]).T
        b -= alpha * ui[0]; omega[0] -= alpha * ui[1]; omega[1] -= alpha * ui[2]
        t = (omega * x + b)
        Loss = np.array(1 - np.multiply(y, t))
        Loss *= (Loss > 0)
        # Loss += max(0, h[0, 0])
        J.append(np.sum(Loss)/n + lamda/2 * np.dot(omega, omega))

print("Parameters of estimated hyperplane are: b = ", b[0,0], "w_1 = ", omega[0], "w_2 = ",
    omega[1])
print("the minimum achieved value of the objective function is: ", min(J))
t = pl.frange(0,8,0.01)
n = pl.frange(1,Iteration*n)
plt.plot(t, -omega[0]/omega[1] * t - b[0,0]/omega[1], label = 'learned line')
legend = plt.legend(loc='upper right', shadow=True)

# Plot the objective function
plt.figure()
plt.plot(n, J, label = 'Objective Function')
plt.show()
```

The result is:

```
Parameters of estimated hyperplane are: b = -0.9463755603126593 w_1 = -2.200214343271486 w_2
    = 11.238670609411892
the minimum achieved value of the objective function is: 0.29596596817675697
```

The plots showing the data and the learned line, and showing J as a function of iteration number is as bellow:





(d)

According to the two plots of objective value above, we can see that with same iteration times, clearly stochastic subgradient method converges faster than subgradient method. This is because subgradient method focuses on the global gradient that can let parameters iterate straightly to the steepest descent direction, while stochastic subgradient method focuses on the descent direction of current data $i(i \in 1, \cdots, N)$. For one iteration time, subgradient method only take one step to optimal solution but stochastic subgradient has taken N steps to optimal solution where N is the size of data. It's reasonable that SGD converges faster.