HW4_P1_3

October 14, 2018

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import csv
# load feature variables and their names
X = np.loadtxt("nuclear_x.csv", delimiter=",", skiprows=1)
with open("nuclear_x.csv", "r") as f:
    X_colnames = next(csv.reader(f))
# load salaries
y = np.loadtxt("nuclear_y.csv", delimiter=",", skiprows=1)
```

In [2]:
```
def GD(X, y, Lambda, theta):
    n = len(X)
    soft_margin = np.zeros(n)
    gradient = np.zeros(len(X[0])+1)
    b = theta[0]
    w = theta[1:]
    for i in range(len(X)):
        soft_margin[i] = 1 - y[i] * (np.dot(w, X[i,:]) + b)

    soft_margin[soft_margin < 0] = 0
    soft_margin[soft_margin > 0] = 1

    gradient[0] = (-1/n) * sum(soft_margin*y)
    gradient[1:] = np.array([sum(soft_margin*(1/n)*(-y*X[:,0] + Lambda*w[0])), sum(sof

    return gradient
```

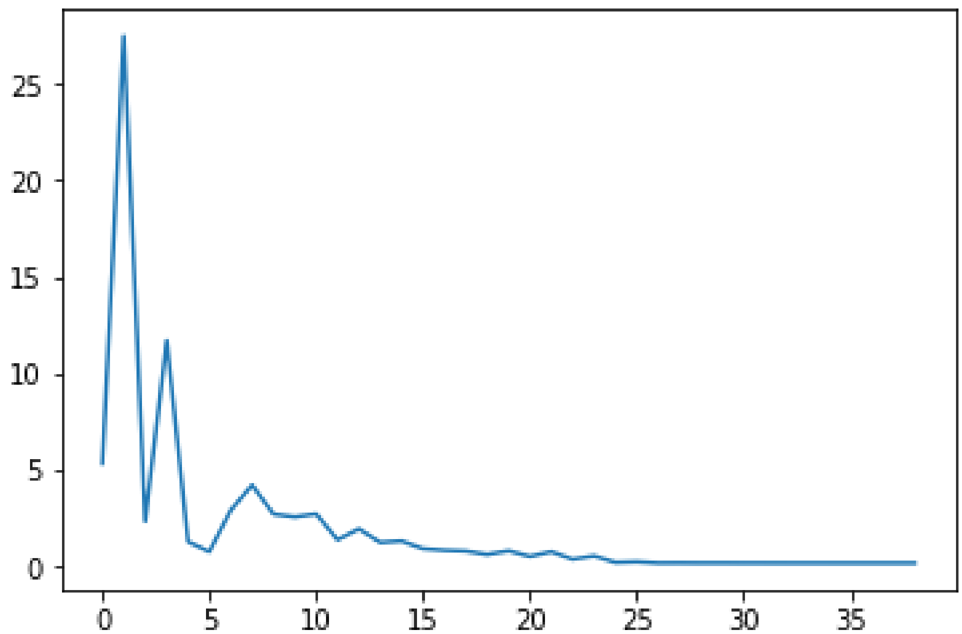
In [3]:
```
def checkObjective(X, y, Lambda, theta):
    n = len(X)
    Loss_i = np.zeros(n)
    b = theta[0]
    w = theta[1:]
    for i in range(len(X)):
        Loss_i[i] = (1/n) * max(0, 1 - y[i] * (np.dot(w, X[i,:]) + b))
    Loss = sum(Loss_i) + (Lambda/2)*np.linalg.norm(theta[1:])
    return Loss
```

In [4]:
```
# Problem1-3 gradient decent
#np.random.seed(0)
```

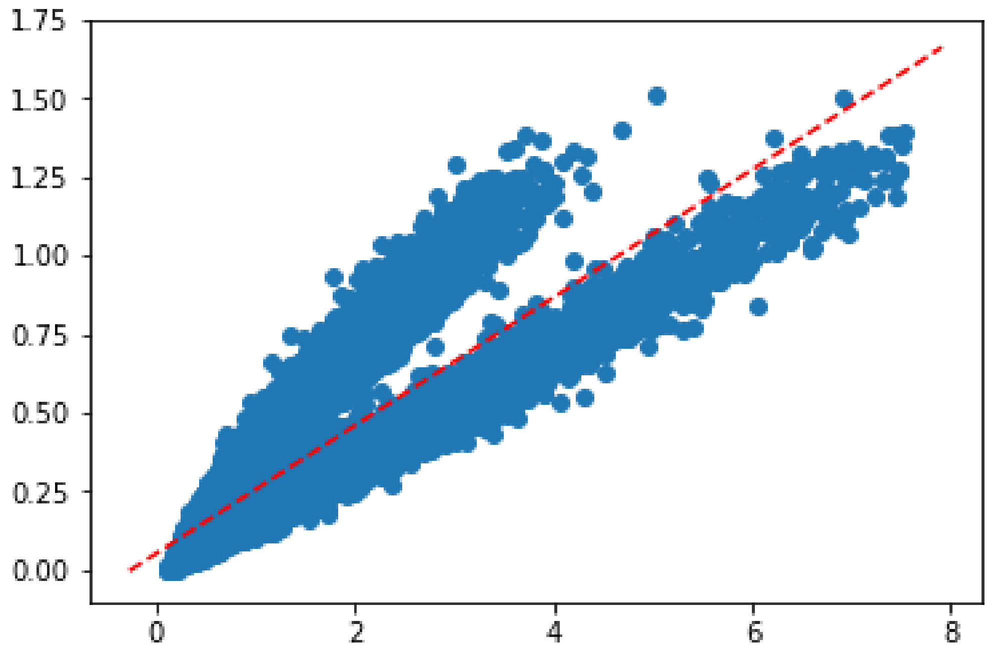
```
theta_0_GD = np.random.rand(len(X[0])+1) # theta = [b w].T

steps = 40
Loss_GD = np.zeros(steps-1)
Lambda = 0.001
theta_GD = theta_0_GD
for k in range(1,steps):
    theta_GD = theta_GD - (100/k) * GD(X, y, Lambda, theta_GD) # theta_k+1 = theta_k -
    #print(theta)
    Loss_GD[k-1] = checkObjective(X, y, Lambda, theta_GD)
    #print(Loss)
```

In [5]:
```
plt.plot(Loss_GD)
plt.show()
```



In [18]:
```
plt.scatter(X[:,0],X[:,1])
slope = -theta_GD[1] / theta_GD[2]
axes = plt.gca()
x_vals = np.array(axes.get_xlim())
y_vals = slope * x_vals + (-theta_GD[0]/theta_GD[2])
plt.plot(x_vals, y_vals, '--', color='red')
plt.show()
```



In [20]: theta_GD

Out[20]: array([-1.08775537, -4.20154906, 20.65468971])

In [21]: Loss_GD[-1]

Out[21]: 0.17706469002736333

In [9]:
```
def SGM(X, y, Lambda, theta):
    n = len(X)
    soft_margin = np.zeros(n)
    gradient = np.zeros((len(X), len(X[0])+1))
    b = theta[0]
    w = theta[1:]
    order = np.random.permutation(n)
    for i in range(len(X)):
        soft_margin[order[i]] = 1 - y[order[i]] * (np.dot(w, X[order[i],:]) + b)
    soft_margin[soft_margin < 0] = 0
    soft_margin[soft_margin > 0] = 1

    gradient[:,0] = (-1/n) * soft_margin * y
    gradient[:,1:] = np.array([soft_margin*(1/n)*(-y*X[:,0] + Lambda*w[0]), soft_margi

    return gradient
```

In [10]:
```
# Problem1-3 stochastic gradient method
#np.random.seed(0)
```

```python
theta_0 = np.random.rand(len(X[0])+1) # theta = [b w].T

n = len(X)
it = 40
steps = it*n
Loss = []#np.zeros(steps-1)
Lambda = 0.001
theta = theta_0
for k in range(1,it):
    #if (k-1) % n == 0:
    J_temp = SGM(X, y, Lambda, theta)
    for i in range(n):
        theta = theta - (100/k) * J_temp[i,:] # theta_k+1 = theta_k - alpha_k * gi(th
        #theta = theta - (100/((k-1)/n + 1)) * J_temp[i,:] # theta_k+1 = theta_k - al
        #print(theta)
        #Loss[k-1+i] = checkObjective(X, y, Lambda, theta)
        #print(k-1+i)
        Loss.append(checkObjective(X, y, Lambda, theta))
        #print(Loss)
        if (i+1) % 5000 == 0:
            print('[k, i]: ', [k, i])
            print(Loss[-1])
```

```
[k, i]:  [1, 4999]
13.279524368150613
[k, i]:  [1, 9999]
26.807850824331823
[k, i]:  [1, 14999]
40.4911721614336
[k, i]:  [2, 4999]
43.97356389907819
[k, i]:  [2, 9999]
34.36788320413823
[k, i]:  [2, 14999]
24.936821510208464
[k, i]:  [3, 4999]
8.954290021813433
[k, i]:  [3, 9999]
2.659479570299297
[k, i]:  [3, 14999]
3.214858914877923
[k, i]:  [4, 4999]
5.028651374763038
[k, i]:  [4, 9999]
1.165575692847243
[k, i]:  [4, 14999]
2.8716606433199634
[k, i]:  [5, 4999]
```

```
3.2635207965341224
[k, i]:  [5, 9999]
0.32822319073847506
[k, i]:  [5, 14999]
3.5216439416711287
[k, i]:  [6, 4999]
4.339838850788406
[k, i]:  [6, 9999]
1.7669862461527068
[k, i]:  [6, 14999]
0.596227680674119
[k, i]:  [7, 4999]
0.6723870412813392
[k, i]:  [7, 9999]
1.4582160976975491
[k, i]:  [7, 14999]
3.8071830229844523
[k, i]:  [8, 4999]
4.295099744333947
[k, i]:  [8, 9999]
2.3647773927827025
[k, i]:  [8, 14999]
0.5912820301641355
[k, i]:  [9, 4999]
0.46615543154865424
[k, i]:  [9, 9999]
1.8004277705526812
[k, i]:  [9, 14999]
3.389676070713953
[k, i]:  [10, 4999]
3.457785547008994
[k, i]:  [10, 9999]
1.9235246843598939
[k, i]:  [10, 14999]
0.5873580241484467
[k, i]:  [11, 4999]
0.3842253172515912
[k, i]:  [11, 9999]
0.861884341124143
[k, i]:  [11, 14999]
1.7362774707033133
[k, i]:  [12, 4999]
1.4282384097465008
[k, i]:  [12, 9999]
0.4237245678652429
[k, i]:  [12, 14999]
0.6859705264788962
[k, i]:  [13, 4999]
```

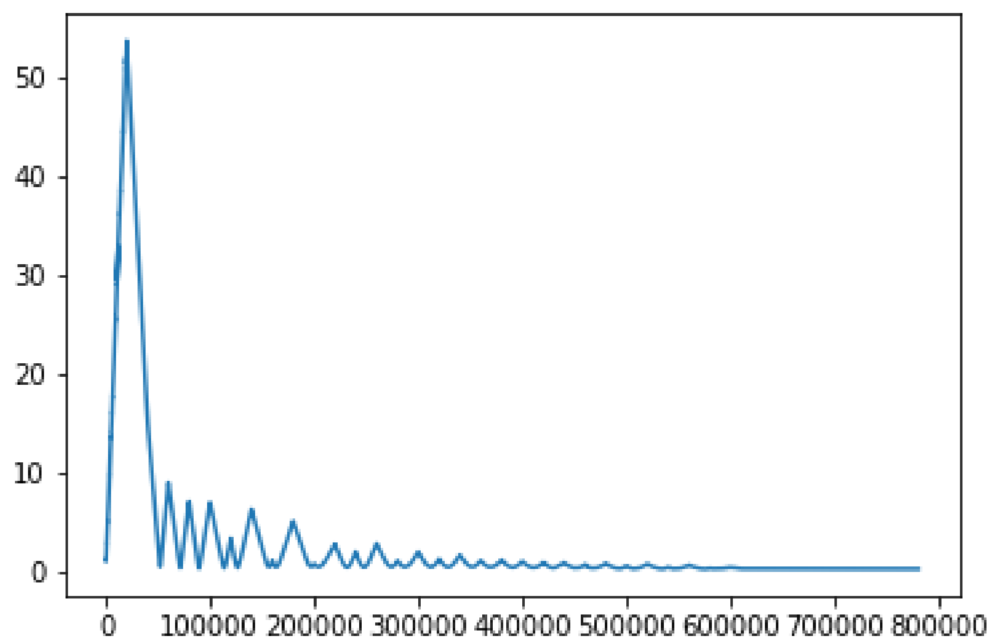
```
0.6567857427613332
[k, i]:  [13, 9999]
0.47421926661101305
[k, i]:  [13, 14999]
1.4517668703693538
[k, i]:  [14, 4999]
1.6482531975261934
[k, i]:  [14, 9999]
0.7096476610393796
[k, i]:  [14, 14999]
0.40417922645165844
[k, i]:  [15, 4999]
0.44708696918973206
[k, i]:  [15, 9999]
0.5031216419862754
[k, i]:  [15, 14999]
1.0780394416955312
[k, i]:  [16, 4999]
1.013892019910262
[k, i]:  [16, 9999]
0.4382124836477374
[k, i]:  [16, 14999]
0.5172913237537052
[k, i]:  [17, 4999]
0.5421300497809156
[k, i]:  [17, 9999]
0.40633174758754753
[k, i]:  [17, 14999]
0.8390621395465658
[k, i]:  [18, 4999]
0.8560569109085975
[k, i]:  [18, 9999]
0.4164745971110191
[k, i]:  [18, 14999]
0.49108835117398164
[k, i]:  [19, 4999]
0.5338269991607183
[k, i]:  [19, 9999]
0.3676998596953023
[k, i]:  [19, 14999]
0.5893453438485156
[k, i]:  [20, 4999]
0.5740443034282295
[k, i]:  [20, 9999]
0.34101428393403743
[k, i]:  [20, 14999]
0.5089916532415247
[k, i]:  [21, 4999]
```

```
0.5188844514733386
[k, i]:  [21, 9999]
0.32062062782696704
[k, i]:  [21, 14999]
0.4424799440880291
[k, i]:  [22, 4999]
0.43314171694216363
[k, i]:  [22, 9999]
0.2939234621243925
[k, i]:  [22, 14999]
0.46759114772785393
[k, i]:  [23, 4999]
0.4727732413328101
[k, i]:  [23, 9999]
0.28437722461847337
[k, i]:  [23, 14999]
0.32656960833809257
[k, i]:  [24, 4999]
0.31380916111074536
[k, i]:  [24, 9999]
0.26613020771696994
[k, i]:  [24, 14999]
0.4433367917419133
[k, i]:  [25, 4999]
0.44224406597001503
[k, i]:  [25, 9999]
0.2645622267278133
[k, i]:  [25, 14999]
0.24726481095855712
[k, i]:  [26, 4999]
0.24073015567868047
[k, i]:  [26, 9999]
0.2510865479441863
[k, i]:  [26, 14999]
0.41596821012264884
[k, i]:  [27, 4999]
0.4157308437552489
[k, i]:  [27, 9999]
0.2543506194478844
[k, i]:  [27, 14999]
0.2047645528014371
[k, i]:  [28, 4999]
0.20513487957904813
[k, i]:  [28, 9999]
0.23284316994212392
[k, i]:  [28, 14999]
0.3650380297152208
[k, i]:  [29, 4999]
```

```
0.36886444502017757
[k, i]:  [29, 9999]
0.24331163657771515
[k, i]:  [29, 14999]
0.18757393300399186
[k, i]:  [30, 4999]
0.19177674728854976
[k, i]:  [30, 9999]
0.20283059735719802
[k, i]:  [30, 14999]
0.2711855009423415
[k, i]:  [31, 4999]
0.27717031266119313
[k, i]:  [31, 9999]
0.21608496152858161
[k, i]:  [31, 14999]
0.1845001107839231
[k, i]:  [32, 4999]
0.18593491524084577
[k, i]:  [32, 9999]
0.18405926340665654
[k, i]:  [32, 14999]
0.19700569535929385
[k, i]:  [33, 4999]
0.19591232290760038
[k, i]:  [33, 9999]
0.1846218740939257
[k, i]:  [33, 14999]
0.18366874949503395
[k, i]:  [34, 4999]
0.18568301643603796
[k, i]:  [34, 9999]
0.18284077042986313
[k, i]:  [34, 14999]
0.19029314920539947
[k, i]:  [35, 4999]
0.1897871178454248
[k, i]:  [35, 9999]
0.18341775251732098
[k, i]:  [35, 14999]
0.18331063887484317
[k, i]:  [36, 4999]
0.18492971390556306
[k, i]:  [36, 9999]
0.18260292957808041
[k, i]:  [36, 14999]
0.18622271632032997
[k, i]:  [37, 4999]
```

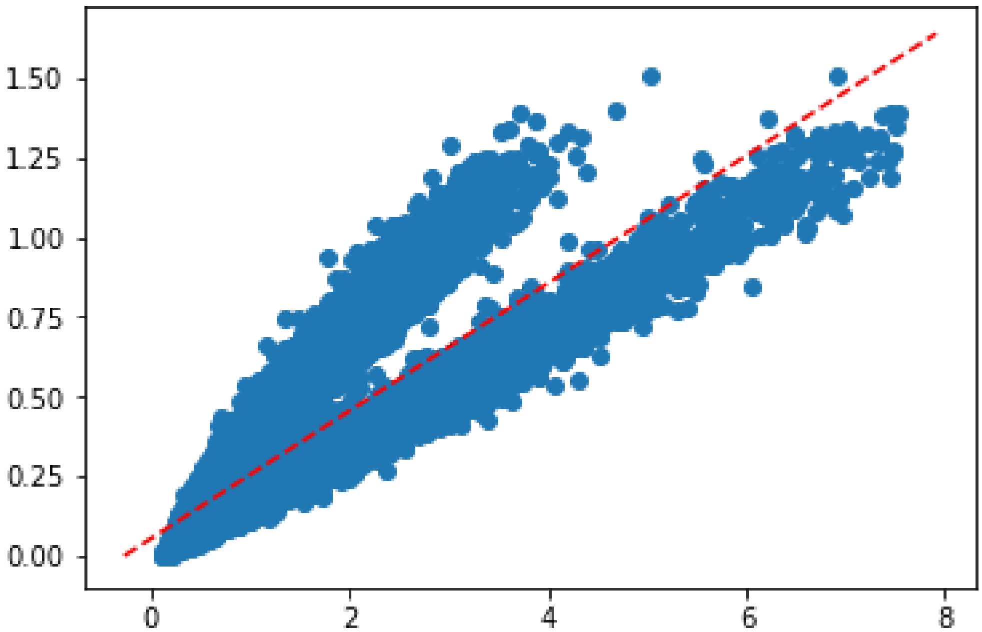
```
0.18616001630019124
[k, i]:  [37, 9999]
0.18291254323409156
[k, i]:  [37, 14999]
0.18300437887522852
[k, i]:  [38, 4999]
0.18405721771157896
[k, i]:  [38, 9999]
0.18257701436724272
[k, i]:  [38, 14999]
0.1840096686978361
[k, i]:  [39, 4999]
0.18393771603572798
[k, i]:  [39, 9999]
0.18267584797953823
[k, i]:  [39, 14999]
0.18275942888767868
```

In [11]:
```python
plt.plot(Loss)
plt.show()
```



In [19]:
```python
plt.scatter(X[:,0],X[:,1])
slope = -theta[1] / theta[2]
axes = plt.gca()
x_vals = np.array(axes.get_xlim())
```

```
y_vals = slope * x_vals + (-theta[0]/theta[2])
plt.plot(x_vals, y_vals, '--', color='red')
plt.show()
```




```
In [13]: print([X[index_min,0], X[index_max,0]], [y1, y2])

[0.38204828657061907, 3.2769927100934244] [-0.003303984298747231, 7.531590858610133]

In [14]: theta

Out[14]: array([-0.97315986, -3.62502033, 18.0781542 ])

In [22]: Loss[-1]

Out[22]: 0.1846215997454534

In [ ]:
```