Homework 1

Jingwei Zhang 201528013229095

2015-11-16

1 Problem 1

2 Problem 2

3 Problem 3

4 Problem 4

5 Programming Problem 1

5.1 Result

The Number of Iterations: 23 for ω_1 and ω_2, 16 for ω_3 and ω_2.

5.2 Code

```
#!/usr/bin/python3
# coding=utf-8

import numpy as np
import matplotlib.pyplot as plt

# constant values
omega1 = np.array([
                    [0.1, 1.1],
                    [6.8, 7.1],
                    [-3.5, -4.1],
                    [2.0, 2.7],
                    [4.1, 2.8],
                    [3.1, 5.0],
                    [-0.8, -1.3],
                    [0.9, 1.2],
                    [5.0, 6.4],
                    [3.9, 4.0]
                    ])
omega2 = np.array([
                    [7.1, 4.2],
                    [-1.4, -4.3],
                    [4.5, 0.0],
                    [6.3, 1.6],
                    [4.2, 1.9],
                    [1.4, -3.2],
                    [2.4, -4.0],
                    [2.5, -6.1],
```

```python
                    [8.4, 3.7],
                    [4.1, -2.2]
                    ])
omega3 = np.array([
                    [-3.0, -2.9],
                    [0.5, 8.7],
                    [2.9, 2.1],
                    [-0.1, 5.2],
                    [-4.0, 2.2],
                    [-1.3, 3.7],
                    [-3.4, 6.2],
                    [-4.1, 3.4],
                    [-5.1, 1.6],
                    [1.9, 5.1]
                    ])
origin = [0,0,0]

def eta(k):
    return 1

def batch_perception(a, b, start_a):
    s = [[1,x[0],x[1]] for x in a]
    s.extend([[-1,-x[0],-x[1]] for x in b])
    s = np.array(s)
    a = start_a
    Y = s[np.inner(a,s) <= 0]
    iter_cnt = 0
    while(len(Y) > 0):
        iter_cnt += 1
        a += eta(iter_cnt) * np.sum(Y, axis=0)
        Y = s[np.inner(a,s) <= 0]
    return a,iter_cnt

if __name__ == '__main__':
    # Problem a)
    a_a, cnt_a = batch_perception(omega1, omega2, origin)
    print(a_a,cnt_a)

    # Problem b)
    a_b, cnt_b = batch_perception(omega3, omega2, origin)
    print(a_b,cnt_b)
```

6 Programming Problem 2

6.1 Result

6.2 Code

```
#!/usr/bin/python3
# coding=utf-8

import numpy as np
import matplotlib.pyplot as plt

# constant values
omega1 = np.array([
                    [0.1, 1.1],
                    [6.8, 7.1],
                    [-3.5, -4.1],
                    [2.0, 2.7],
                    [4.1, 2.8],
                    [3.1, 5.0],
                    [-0.8, -1.3],
                    [0.9, 1.2],
                    [5.0, 6.4],
                    [3.9, 4.0]
                    ])
omega2 = np.array([
                    [7.1, 4.2],
                    [-1.4, -4.3],
                    [4.5, 0.0],
                    [6.3, 1.6],
                    [4.2, 1.9],
                    [1.4, -3.2],
                    [2.4, -4.0],
                    [2.5, -6.1],
                    [8.4, 3.7],
                    [4.1, -2.2]
                    ])
```

```
omega3 = np.array([
                   [-3.0, -2.9],
                   [0.5, 8.7],
                   [2.9, 2.1],
                   [-0.1, 5.2],
                   [-4.0, 2.2],
                   [-1.3, 3.7],
                   [-3.4, 6.2],
                   [-4.1, 3.4],
                   [-5.1, 1.6],
                   [1.9, 5.1]
                   ])
omega4 = np.array([
                   [-2.0, 8.4],
                   [-8.9, 0.2],
                   [-4.2, 7.7],
                   [-8.5, -3.2],
                   [-6.7, -4.0],
                   [-0.5, -9.2],
                   [-5.3, -6.7],
                   [-8.7, -6.4],
                   [-7.1, -9.7],
                   [-8.0, -6.3]
                   ])
origin = [0,0,0]

def eta(k):
    return 0.5

def rand_start(n):
    return [np.random.uniform() for i in range(n)]

def get_init_b(n):
    return [np.random.uniform() for i in range(n)]

def Ho_Kashyap(s1, s2, a_start, b_start, k_max, b_min):
    s = [[1,x[0],x[1]] for x in s1]
    s.extend([[-1,-x[0],-x[1]] for x in s2])
    Y = np.matrix(s)
    Y_inv = np.linalg.pinv(Y)
    a = np.matrix(a_start).T
    b = np.matrix(b_start).T
    all_e = []
    split_able = False
    for k in range(1, k_max+1):
        # inpu = input("interation")
        e = Y * a - b
        # all_e.append((e.T * e)[0,0])
        cnt = 0
        for i in range(len(s)):
            if (a.T * ((Y[i,]).T))[0, 0] < 0:
                cnt += 1
        all_e.append(cnt)
        # print(cnt)
        if(np.max(np.abs(e)) < b_min):
            split_able = True
```

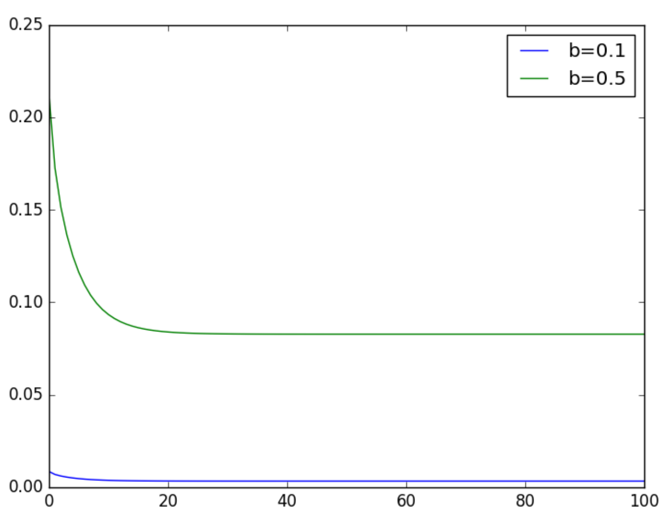
```
                break
            e_plus = e
            e_plus[e < 0] = 0
            b = b + (2 * eta(k)) * e_plus
            a = Y_inv * b
    print(split_able)
    return all_e

if __name__ == '__main__':
    k_max = 10000
    start_a = rand_start(3)
    start_b = get_init_b(len(omega1)+len(omega3))
    b_min = 1E-5
    # omega 1 and omega 3
    x = [i for i in range(1, k_max + 1)]
    all_E_1 = Ho_Kashyap(omega1, omega3,
        start_a, start_b, k_max, b_min)
    # print(all_E_1[len(all_E_1)-1])
    # omega 2 and omega 4
    all_E_2 = Ho_Kashyap(omega2, omega4,
        start_a, start_b, k_max, b_min)
    # print(all_E_2)
    # plot
    ax = plt.gca()
    ax.set_xscale('log')
    plt.title("Training Errors")
    plt.plot(x,all_E_1,color='b',
        label="omega 1 and omega 3")
    plt.plot(x,all_E_2,color='r',
        label="omega 2 and omega 4")
    plt.legend()
    plt.show()
```

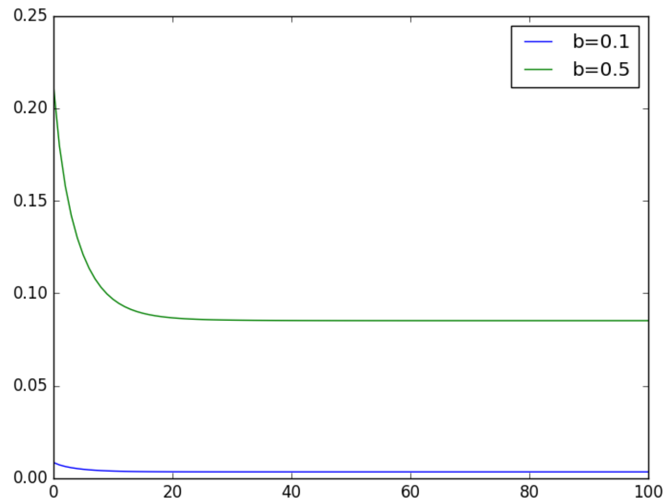
7 Programming Problem 3

7.1 Result

Batch Relaxation:



Single Sample Relaxation:



7.2 Code

```
#!/usr/bin/python3
# coding=utf-8

import numpy as np
import matplotlib.pyplot as plt

# constant values
omega1 = np.array([
    [0.1, 1.1],
    [6.8, 7.1],
    [-3.5, -4.1],
    [2.0, 2.7],
    [4.1, 2.8],
    [3.1, 5.0],
    [-0.8, -1.3],
    [0.9, 1.2],
    [5.0, 6.4],
    [3.9, 4.0]
])
omega2 = np.array([
    [7.1, 4.2],
    [-1.4, -4.3],
    [4.5, 0.0],
    [6.3, 1.6],
    [4.2, 1.9],
    [1.4, -3.2],
    [2.4, -4.0],
    [2.5, -6.1],
    [8.4, 3.7],
    [4.1, -2.2]
])
omega3 = np.array([
    [-3.0, -2.9],
```

```
    [0.5, 8.7],
    [2.9, 2.1],
    [-0.1, 5.2],
    [-4.0, 2.2],
    [-1.3, 3.7],
    [-3.4, 6.2],
    [-4.1, 3.4],
    [-5.1, 1.6],
    [1.9, 5.1]
])
omega4 = np.array([
    [-2.0, 8.4],
    [-8.9, 0.2],
    [-4.2, 7.7],
    [-8.5, -3.2],
    [-6.7, -4.0],
    [-0.5, -9.2],
    [-5.3, -6.7],
    [-8.7, -6.4],
    [-7.1, -9.7],
    [-8.0, -6.3]
])
origin = [0, 0, 0]
k_max = 100

def eta(k):
    return 0.1

def judge_function(Y, a, b):
    return 1.0 / 2 * np.sum([(np.inner(a, y) - b)**2 / np.inner(y, y)
        for y in Y])

def coff(y, a, b):
    return (b - np.inner(a, y)) / np.inner(y, y)

def batch_relaxition(s1, s2, start_a, b):
    s = [[1, x[0], x[1]] for x in s1]
    s.extend([[-1, -x[0], -x[1]] for x in s2])
    s = np.array(s)
    a = start_a
    Y = s[np.inner(a, s) <= b]
    J = [judge_function(Y, a, b)]
    k = 0
    while(len(Y) > 0 and k < k_max):
        # inp = input("iteration"+str(len(Y)))
        y = Y[0]
        # print(y, coff(y, a, b)*y)
        k += 1
        a += eta(k) * np.sum([coff(y, a, b) * y for y in Y], axis=0)
        Y = s[np.inner(a, s) <= b]
        J.append(judge_function(Y, a, b))
    return J

def single_sample_relaxition(s1, s2, start_a, b):
    s = [[1, x[0], x[1]] for x in s1]
    s.extend([[-1, -x[0], -x[1]] for x in s2])
```

```python
    s = np.array(s)
    a = start_a
    Y = s[np.inner(a, s) <= b]
    J = [judge_function(Y, a, b)]
    k = 0
    while(len(Y) > 0 and k < k_max):
        k += 1
        for y in s:
            a += eta(k) * coff(y, a, b) * y
        Y = s[np.inner(a, s) <= b]
        J.append(judge_function(Y, a, b))
    return J

if __name__ == '__main__':
    s1 = omega1
    s2 = omega3
    ax = plt.gca()
    # ax.set_xscale('log')
    J_1 = batch_relaxition(s1, s2, origin, 0.1)
    x = [i for i in range(len(J_1))]
    plt.plot(x, J_1, label="b=0.1")
    J_2 = batch_relaxition(s1, s2, origin, 0.5)
    x = [i for i in range(len(J_2))]
    plt.plot(x, J_2, label="b=0.5")
    plt.legend()
    plt.show()

    ax = plt.gca()
    # ax.set_xscale('log')
    J_1 = single_sample_relaxition(s1, s2, origin, 0.1)
    x = [i for i in range(len(J_1))]
    plt.plot(x, J_1, label="b=0.1")
    J_2 = single_sample_relaxition(s1, s2, origin, 0.5)
    x = [i for i in range(len(J_2))]
    plt.plot(x, J_2, label="b=0.5")
    plt.legend()
    plt.show()
```