# CS589 Machine Learning
# Homework 4

Submitted by- Ravi Agrawal Due on- Nov 21nd, 2017

**A 1:**

```
def p_j_alpha(j, alpha):
    X_i = 1.0
    if j[0] !=0:
        X_i = 1e-80
    else:
        for i in range(1, len(j)):
            X_i = X_i*alpha if j[i] == j[i-1] else X_i*(1.0 - alpha)
    return X_i
```

**A 3:**

Case 3 : Input : J=(1,1,0,1,0,1) , α=0.2, Output= **1e-80 or 0.0**
Case 4 : Input : J=(0,1,0,1,0,0) , α=0.2, Output=**0.0819**

**B 1:**

```
def p_b_j(b, j, dic):
    X_i = 1.0
    for i in range(len(b)):
        X_i *= dic[j[i], b[i]]
    return X_i
```

**B 2:**

Case 3 : Input : J=(0,1,1,0,0,1), B = (1,0,1,1,1,0), Output = **0.04147**
Case 4 : Input : J=(1,1,0,0,1,1), B = (0,1,1,0,1,1), Output = **0.00014**

**C 1:**

```
def p_alpha(alpha):
    if alpha >= 0.0 and alpha <= 1.0:
        return 1.0
    else:
        return 1e-80
```

**D 1:**

```
def p_alpha_j_b(b, j, alpha):
    return p_alpha(alpha)*p_b_j(b, j, dic)*p_j_alpha(j, alpha)
```

**D 3:**

Case 3 : Input : J=(0,0,0,0,0,1), B = (0,1,1,1,0,1), α=0.63, Output= **0.00011**
Case 4 : Input : J=(0,0,1,0,0,1,1), B = (1,1,0,0,1,1,1), α=0.23, Output=**5.1191e-06**

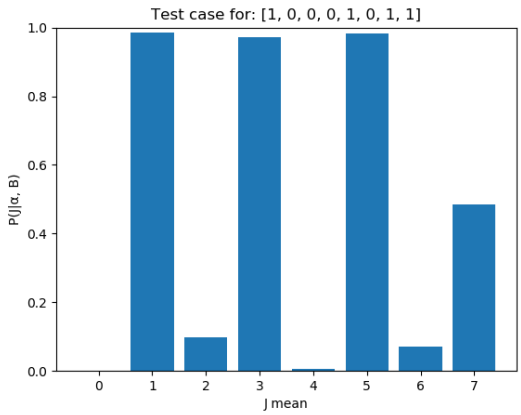
**E :**

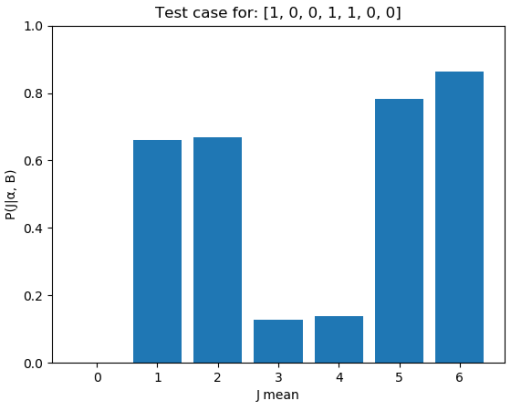
```
def random_j(j):
    length  = len(j)
    index = np.random.randint(0, length)
    j_new = np.copy(j)
    j_new[index] = int(not j_new[index])
    return j_new
```

**F 1:**

```
def p_j_alpha_b(b, alpha, iteration):
    j = np.array([0 for i in range(len(b))])
    j_mean = np.array([0 for i in range(len(b))])
    for i in range(iteration):
        j_new = np.array(random_j(j))
        acceprance_ratio = p_alpha_j_b(b, j_new, alpha)*1.0/p_alpha_j_b(b, j, alpha)*1.0
        if random.random() <= acceprance_ratio:
            j = j_new
        j_mean += j
    return j_mean/(1.0*iteration)
```

**F 3:**




**F 4:**

Case 2 :

B=(1,0,0,0,1,0,1,1), α=0.11, Output=**[ 0.0, 0.9778, 0.09, 0.9747, 0.0038, 0.9862, 0.0628, 0.4821]**

Case 3 :

B = (1,0,0,1,1,0,0), α=0.75, Output= **[ 0.0, 0.6442, 0.667, 0.1081, 0.1046, 0.7767, 0.8517]**

**G:**

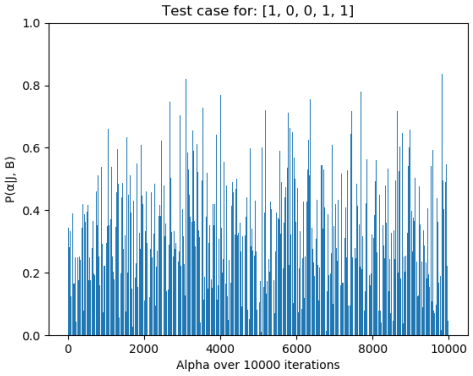
```
def random_alpha(alpha):
```

```
    return np.random.rand()
```

**H 1:**

```
def p_alpha_given_j_b(j, b, iteration):
    alpha_mean =[]
    alpha = random.random()
    for i in range(int(iteration)):
        alpha_new = random_alpha(alpha)
        #print(p_alpha_j_b(b, j, alpha))
        acceprance_ratio = p_alpha_j_b(b, j, alpha_new)/p_alpha_j_b(b, j, alpha)
        if random.random() <= acceprance_ratio:
            alpha = alpha_new
        alpha_mean.append(alpha)
    return sum(alpha_mean)/(1.0*iteration)
```

**H 3:**




**H 4:**

Case 4 : Input : J=(0,1,1,1,1,1,1,0), B = (1,0,0,1,1,0,0,1), Output=0.6649
Case 5 : Input : J=(0,1,1,0,1,0), B = (1,0,0,1,1,1), Output=0.2900

**I:**

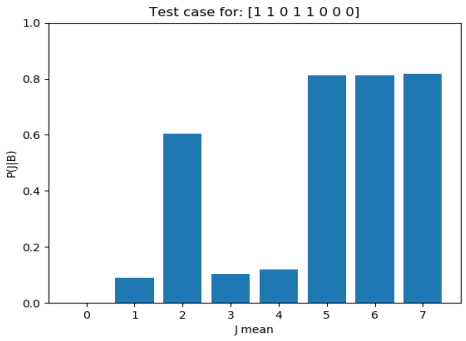
```
def proposal_alpha_j(j, alpha):
    return random_j(j), random_alpha(alpha)
```

**J 1:**

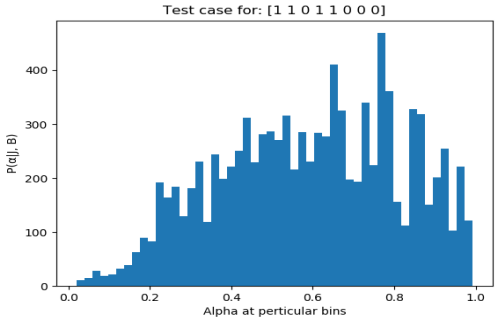
```
def p_new_alpha_new_j_b(b, iteration):
    alpha_mean = 0.0
    alpha = random.random()
    j = np.array([0 for i in range(len(b))])
    j_mean = np.array([0 for i in range(len(b))])
```

```
    for i in range(int(iteration)):
        alpha_new = random_alpha(alpha)
        j_new = np.array(random_j(j))
        acceprance_ratio = p_alpha_j_b(b, j_new, alpha_new)/p_alpha_j_b(b, j, alpha)
        if random.random() <= acceprance_ratio:
            alpha = alpha_new
            j = j_new
        alpha_mean += alpha
        j_mean += j
    return j_mean/(1.0*iteration), alpha_mean/(1.0*iteration)
```

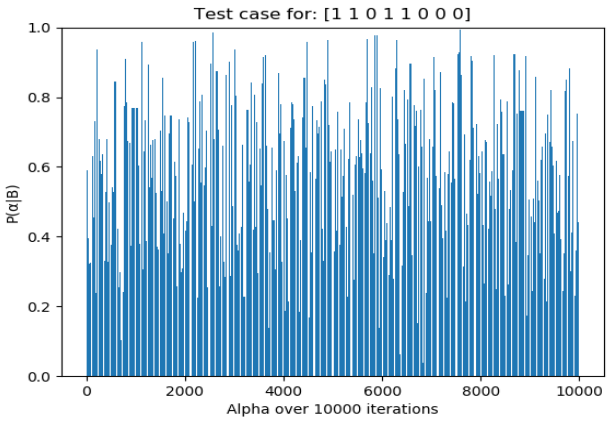
**J 2 1:**




**J 2 2:**




**J 2 3:**




**J 3:** The alpha (running average) starts to converge by the end of the 10,000 iterations. By looking at the second graph it gets clear that alpha is following the normal distribution with average being right skewed, The lower and higher value of the alpha are having selected much less number of times then the values of alpha at 1.5 standard deviation of the mean of the alpha.

**K 1:**

```
def p_n_1_given_alpha(j_n, alpha):
    if j_n == 0:
        return alpha*0.8 + (1 - alpha)*0.1
```

```
    if j_n == 1:
        return alpha*0.1 + (1 – alpha)*0.8
```

**K 2:**

Case 3 : Input : α = 0.33456, J n = 0, Output=0.33419

Case 4 : Input : α = 0.5019, J n = 1,Output=0.44867

**L 1:**

```
def p_j_alpha_given_b(b, iteration):
    j_proposed, alpha_proposed = p_new_alpha_new_j_b(b, iteration)
    j = 1*(j_proposed > 0.5)
    return p_n_1_given_alpha(j[-1], alpha_proposed)
```

**M 1 (USERNAME – IMRAVIAGRAWAL):** The MSE on the kaggle is 0.00321, I have have ran model for the 10,000 iteration and then averaged over the all values to get the best value.