1- option 2 using the formula given in page 3, lecture 8, the correct answer is: 0.05

In [2]:

```
import numpy as np
import math
def binom(n, k):
    #computes the binomial coefficient
    return math.factorial(n) // math.factorial(k) // math.factorial(n - k)
L=15
eps=0.3
sum_prob=0
#sums up the probability of the cases where more than half of the base learers a
re wrong
for k in range(math.ceil(L/2),L+1):
    sum_prob+=(binom(L, k)*np.power(eps,k)*np.power(1-eps,L-k))
print("Probability that the ensemble makes an incorrect prediction is:",np.round
(sum_prob,4))
```

```
Probability that the ensemble makes an incorrect prediction is: 0.05
```

2- option 2 In AdaBoost algorithm page 20, lecture 8, in order to obtain positive weights(page 20, line 5) for a base learner, the corresponding weighted error should be less than 0.5 (random guessing).

3- option 1

AdaBoost minimizes exponential loss, whereas LogitBoost applies logistic loss as the surrogates for zero-one loss.

4- option 3

In [8]:

```
import numpy as np
x1 = np.array([.1,.2,.4,.8, .8, .05,.08,.12,.33,.55,.66,.77,.22,.2,.3,.6,.5,.6,.
25,.3,.5,.7,.6])
x2 = np.array([.2,.65,.7,.6, .3,.1,.4,.66,.22,.65,.68,.55,.44,.1,.3,.4,.3,.15,.1
5,.5,.55,.2,.4])
N=np.shape(x1)
labels = np.array([1,1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1])
X = np.vstack((x1,x2,np.ones(N))).T
def perceptron (X,labels,sample_weight):
    #Inputs:
    #X:  a 2d array, each row represents an example of the training set
    #labels: vector of the examples labels
    #sample_weight: vector of examples weights given by Adaboost
    #Output:
    #pred_labels: the label predicted for each example
    d = np.shape(X)[1]
    w = np.zeros(d)
    i = 1
    while(any([element<=0 for element in [labels[ind]*np.dot(w,x) for ind,x in e
numerate(X)] ])):
        #misclassified examples
        mistakes = np.where([element<=0 for element in [labels[ind]*np.dot(w,x)
for ind,x in enumerate(X)] ])[0]
        pairs = zip(mistakes, sample_weight[mistakes])
        sorted_pairs = sorted(pairs, key=lambda t: t[1], reverse = True)
        #use the misclassified example with maximum weight given by Adaboost
        misclass = sorted_pairs[0][0]
        #weight update
        w = w + labels[misclass]*X[misclass]
        #labels prediction
        pred_labels = [1 if x>0 else -1 for x in [np.dot(w,x) for x in X]]
        i +=1
        if (i>201):
            break
    return pred_labels
def AdaBoost(X,y, M):
    N = len(y)
    y_predict_list,  estimator_weight_list, sample_weight_list = [], [],[]
    #Initialize the sample weights
    sample_weight = np.ones(N) / N
    sample_weight_list.append(sample_weight.copy())
    for m in range(M):
        #fit a classifier
        y_predict = perceptron(X,y,sample_weight)
        #misclassifications
        incorrect = (y_predict != y)
        #weighted error
        estimator_error = np.average(incorrect, weights=sample_weight, axis=0)
        #estimator weight
        estimator_weight =   0.5*np.log((1. - estimator_error) / estimator_error
)
        #compute the sample weights
        sample_weight *= np.exp(estimator_weight *-1*np.asarray(y_predict* y)  )
        #normlamize the sample weights to sum up to 1
        sample_weight /= 2* np.power((estimator_error *(1- estimator_error)),0.5
)
        y_predict_list.append(y_predict.copy())
        estimator_weight_list.append(estimator_weight.copy())
        y_predict_mat = np.asarray(y_predict_list)
```

```
        estimator_weight_mat = np.asarray(estimator_weight_list)
        #compute the ensemble prediction
        preds = (np.array([np.sign((y_predict_mat[:,point] * estimator_weight_ma
t).sum()) for point in range(N)]))
        #compute the accuracy
        accuracy = (preds == y).sum() / N
    return accuracy
accuracy = AdaBoost(X,labels, M=5)
print("accuracy: ", np.round(accuracy,3))
```

```
accuracy:  0.783
```