```
1. Write down the training and test errors of the classifiers obtained after t
= 3, 7, 10, 15, 20 rounds of
boosting.
```

In [152]:
```
import numpy as np

train_data = np.loadtxt("pa5train.txt")
test_data = np.loadtxt("pa5test.txt")
dic_data = np.genfromtxt('pa5dictionary.txt',dtype='str')


def class_pos(val):
    return 1 if val == 1 else -1

def get_pro(val, label):
    return class_pos(val) if label > 0 else class_pos(not val)

class Classifier_data:
    feature = -1
    label = 0
    alpha = -1

    def __init__(self,al,fea,lab):
        self.alpha = al
        self.feature = fea
        self.label = lab

def boost(data,weights):
    label = 0; min_err = float('inf'); feat_best = -1

    for i in range(len(data[0])-1):
        error = sum(weights[x] for x in range(len(data)) if
          class_pos(data[x][i])!=data[x][-1])

        if (error < min_err) :
            feat_best = i
            min_err= error
            label = 1
        elif ( 1.0 - error < min_err ) :
            feat_best = i
            min_err = 1.0 - error
            label = -1
    alpha = 0.5*np.log(((1.0-min_err)/min_err))

    for i in range(len(data)):
        precomp = get_pro(data[i][feat_best], label)
        weights[i]=weights[i]*np.exp(-alpha*data[i][-1]*precomp)

    weights = list(map(lambda elm: elm/sum(weights), weights))
    return Classifier_data(alpha,feat_best,label), weights

def classify(email, classifiers):
    return np.sign(sum(elm.alpha*class_pos(email[elm.feature])
      if elm.label > 0 else elm.alpha*class_pos(not email[elm.feature])
      for elm in classifiers))

def boost_tests(d_train, d_test, num):

        weights = [1.0/len(d_train)] * len(d_train)
        classi_cont_list = []

        for _ in range(num):
            res, weights = boost(d_train,weights)
            classi_cont_list.append(res)
        return classi_cont_list
```

In [157]:
```
boost_rounds= [3,4,7,10,15,20]
res_10 = []
for num in boost_rounds:
    res = (boost_tests(train_data, test_data, num))
    if num == 10: res_10 = res
    print("-------------After "+str(num) + " boosting rounds-------------------")
    print("     training error is "+ str(sum(1 for email in train_data
            if classify(email,res)!=email[-1])/float(len(train_data))))
    print("     test error is "+ str(sum(1 for email in test_data
            if classify(email,res)!=email[-1])/float(len(test_data)))+"\n")
```

```
-------------After 3 boosting rounds-------------------
     training error is 0.06444444444444444
     test error is 0.03875968992248062

-------------After 4 boosting rounds-------------------
     training error is 0.051111111111111114
     test error is 0.03875968992248062

-------------After 7 boosting rounds-------------------
     training error is 0.028888888888888888
     test error is 0.031007751937984496

-------------After 10 boosting rounds-------------------
     training error is 0.015555555555555555
     test error is 0.03875968992248062

-------------After 15 boosting rounds-------------------
     training error is 0.0
     test error is 0.023255813953488372

-------------After 20 boosting rounds-------------------
     training error is 0.0
     test error is 0.023255813953488372
```

2. Based on the dictionary file, write down the words corresponding to the weak learners chosen in the
first 10 rounds of boosting.

In [158]:
```
print("-------------Words chosen by weak learners after 10 rounds are ----------
spam = "\tSpam: "+ ', '.join([dic_data[c.feature] for c in res_10 if c.label < 0]
print(spam)
not_spam = "\tNot Spam: "+', '.join([dic_data[c.feature] for c in res_10 if
      c.label > 0]) + "\n"
print(not_spam)
```

```
------------Words chosen by weak learners after 10 rounds are ----------------
        Spam: remove, free, money, click, want
        Not Spam: language, university, linguistic, fax, de
```