4. Boosting I: Weak Learners and Decision Stumps

In [18]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Load the banknote data into a pandas dataframe
fname = r'banknote.data.txt'
bnote = pd.read_csv(fname,header=None)
# peak at the first five rows
bnote.head()
```

Out[18]:

	0	1	2	3	4
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

d)

We use a vary naive way to find the best decision stump, which is to loop over every value in the j^{th} column of the banknote data and check for both S_j^+ and S_j^- until we find the optimal threshold and direction

In [19]:
```
# the decision stump classifier
def stumpclassify(X,dim,sign,thresh):
    n = X.shape[0]
    f_x = np.zeros(n)
    if sign==1:
        f_x = (X.iloc[:,dim]>=thresh)
    elif sign==-1:
        f_x = (X.iloc[:,dim]<thresh)
    return f_x
```

In [26]:
```
# find the best decision stump threshold and sign by looping over all possible va
def threshfind(X,dim):
    y = X.iloc[:,-1]==1
    P_best = 0
    thresh = np.zeros(0)
    sign = np.zeros(0)
    for s in [1,-1]:
        for t in X.iloc[:,dim]:
            P_curr = np.mean(stumpclassify(X,dim,s,t)==y)
            if P_curr>=P_best:
                P_best = P_curr
                thresh = t
                sign = s
    return thresh,sign,P_best
```

In [27]:
```
# find and display the decision stumps and display them for each of the features
thresh = np.zeros(4)
sign = np.zeros(4)
P_correct = np.zeros(4)
print('thresh sign P(correct)')
for j in range(4):
    thresh[j],sign[j],P_correct[j] = threshfind(bnote,j)
    print(thresh[j],sign[j],round(P_correct[j],4))
```

```
thresh sign P(correct)
0.3223 -1.0 0.8535
5.1815 -1.0 0.7055
8.6521 1.0 0.6268
-5.8638 -1.0 0.5627
```

Problem 5: Boosting II: Aggregating Weak Learners

In [34]:
```
from cvxopt import matrix, solvers
# number of weak learners
m = 4
n = bnote.shape[0]
# letting t = a-b, we change max{t} into -min{-a+b}
c = matrix(np.hstack([np.zeros(m),[-1,1]]))
# now we build the matrix M
M = np.zeros((n,m))
for j in range(m):
    M[:,j] = np.asarray(stumpclassify(bnote,j,sign[j],thresh[j])==(bnote.iloc[:,-
# assemble matrix G as [M|]
G = matrix(np.vstack([np.hstack([-M,np.tile([1,-1],(n,1))]),-np.eye(m+2)]))
# h
h = matrix(np.hstack([np.zeros(n),np.zeros(m+2)]))
# A and b specifies the p summing to 1
A = matrix([[1.],[1.],[1.],[1.],[0.],[0.]])
b = matrix([1.])
sol = solvers.lp(c, G, h, A, b,solver="glpk")

print('p(h)=')
print(np.round(p.squeeze(),2))
```

```
p(h)=
[0.33 0.33 0.   0.33]
```

In [35]:
```
# now we build the matrix H and apply the decision rules of boosted classifier
H = np.zeros((n,m))
for j in range(m):
    H[:,j] = np.asarray(stumpclassify(bnote,j,sign[j],thresh[j]))*2-1
p = np.asarray(sol['x'][:4])
P_correct_boosted = np.mean((np.matmul(H,p)>0).squeeze()==(bnote.iloc[:,-1]==1))

print('\nP(err_boosted)')
print(np.round(P_correct_boosted,4))
```

```

P(err_boosted)
0.8994
```

We see the performance indeed inproved with boosting