In [1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
%matplotlib inline
from collections import OrderedDict
```

In [2]:
```
def getdata_CSV(filename):
    return np.genfromtxt(filename,delimiter=",")

data1 = getdata_CSV("credit_Score.csv")
```

Data Preparation

$Removing\ the\ discrete\ features\ and\ splitting\ the\ data\ into\ Train\ and\ Test\ Set$

In [120]:
```
req_cols= [1,2,7,10,13,14,15]
clean_data = data1[:,req_cols]
```

In [121]:
```
count_1 = 0
count_2 = 0
X_Train=[]
X_Test=[]
for row in clean_data:

    if count_1!=200 and count_2 !=200:

        X_Train.append(row)
        if(row[6]==0):
            count_1+=1
        else:
            count_2+=1

    elif count_1<200 and count_2 == 200:

        if row[len(row)-1]==0:

            X_Train.append(row)
            count_1+=1
        else:
            X_Test.append(row)

    elif count_1==200 and count_2 <200:

        if row[len(row)-1]==1:
            X_Train.append(row)
            count_2+=1
        else:
            X_Test.append(row)


    elif count_1==200 and count_2==200 :

        X_Test.append(row)
```

In [122]:
```
X_Train=np.array(X_Train)
Y_Train = np.array(X_Train[:,X_Train.shape[1]-1])


X_Test =np.array(X_Test)
Y_Test = np.array(X_Test[:,X_Train.shape[1]-1])
X_Test = np.delete(X_Test,-1,1)
print "Trainning Samples", X_Train.shape ,"Testing Samples", X_Test.shape
```

```
Trainning Samples (400L, 7L) Testing Samples (253L, 6L)
```

In [54]:
```
def mean_squared_error(y,y_cap):
    """ Computes the mean squared Error"""
    errors = 0
    for i in range(0,len(y)):
        errors = errors + (y_cap[i] -y[i])**2

    return errors.sum()/len(y)
```

1D 2-Class Gaussian Discriminant Analysis

The Model Parameters are μ and σ

Compute the mean and variance of samples with y=0 and y= 1 separately on X_Train

$\mu_j = 1/m_j \sum_{i=1}^{m_j} X^{(i)}$

$\sigma_j^2 = 1/m_j \sum_{i=1}^{m_j} (X^{(i)} - \mu_j)^2$

In [124]:
```
X_0 = np.array([x[0] for x in X_Train if x[len(x)-1]==0])
X_1 = np.array([x[0] for x in X_Train if x[len(x)-1]==1])
print (X_0).shape , (X_1).shape
print X_Test[0][0]
```

```
(200L,) (200L,)
41.42
```

In [125]:
```
def model_parameters(X0,X1,d=1):
    """ Computes mean and variance """
    if d==1:
        meanJ = np.array([np.mean(X0),np.mean(X1)])
        sigmaJ = np.array([np.var(X0),np.var(X1)])
    else:
        meanJ = np.array([np.mean(X0,axis=0),np.mean(X1,axis=0)])
        sigmaJ = np.array([np.cov(X0.T),np.cov(X1.T)])

    return meanJ,sigmaJ

meanJ,sigmaJ = model_parameters(X_0,X_1)
print meanJ,sigmaJ
```

```
[ 29.78775  34.43025] [ 120.65719844  146.39534244]
```

Univariate Gaussian Discriminant

$g_j(X) = -log(\sigma_j) - (X - \mu_j)^2/\sigma_j^2 + log(\alpha_j)$

Mulitvariate Gaussian Discriminant

$$g_j(X) = -log(|\sum_j|) - 1/2(X-\mu_j)^T \sum^{-1}(X-\mu_j) + log(\alpha_j)$$

In [126]:
```
def gaussian_Discriminant(X,mean,sigma,dim=1,prior_prob=0.5):
    """Based on the dim attribute Discriminant functions
    for Univariate and Multivariate Distributions will be computed"""

    g=0.0

    if dim==1:
        g = -np.log(sigma) - ((X - mean)**2/2*(sigma**2)) + np.log(prior_pro
    else:

        t = -np.log ( np.linalg.det(sigma))/2
        s = - 1/2 * np.dot(np.dot((X-mean).T,np.linalg.inv(sigma)),(X-mean))
        g = t+s

    return g

g_0 = gaussian_Discriminant(X_Test[0][0],meanJ[0],sigmaJ[0])
g_1 = gaussian_Discriminant(X_Test[0][0],meanJ[1],sigmaJ[1])
```

In [127]:
```
def predict(X,meanJ,sigmaJ,dim=1):
    """Predicts the label based on the discriminant value based
    on the memberships of all classes and choice the class with
    Highest membership value"""
    y=[]

    for eachx in X:
        g=[]
        for i in range(0,len(meanJ)):
            gd = gaussian_Discriminant(eachx,meanJ[i],sigmaJ[i],dim=dim)

            g.append(gd)
        y.append(np.argmax(g))

    return y

y_Cap = predict(X_Test[:,0],meanJ,sigmaJ)
```

In [59]:
```
def confussion_matrix(y_cap,y,cls=1):
    """ Compute the confussion matrix for the given predicted and actual cla
    a=0
    b=0
    c=0
    d=0
    for i in range(0,len(y)):
        if y[i]==cls and y_cap[i] ==cls:
            a+=1
        elif y[i]!=cls and y_cap[i] ==cls:
            c+=1
        elif y[i]==cls and y_cap[i] !=cls:
            b+=1
        elif y[i]!=cls and y_cap[i] !=cls:
            d+=1
    return float(a),float(b),float(c),float(d)


a,b,c,d = confussion_matrix(y_Cap,Y_Test,0)
print "Confussion Matrix \n", np.matrix([[a,b],[c,d]])
```

In [60]:
```
def evaluate_performance(y_Cap,y,cls):
    """ Using the Confusion matrix , the metric like Accuracy , Precision ,
    a,b,c,d = confussion_matrix(y_Cap,y,cls)
    r =0
    p= 0
    if a+b+c+d!=0:
        print "Accuracy \t:",(a+d)/(a+b+c+d)
    else:
        print "Can't Compute Accurancy"
    if c+d !=0:

        r = (d)/(c+d)
        print "Recall  \t:",r
        print "False Negative \t:",c/(c+d)
    else:
        print "Can't Compute Recall and False Negative"
        r=0
    if b+d!=0:
        p = d/(b+d)
        print "Precision \t:",p
    else:
        print "Can't Compute Precision "
        p=0
    if a+b!=0:
        print "False Positive \t:",b/(a+b)
        print "True Negative \t:",a/(a+b)
    else:
         print "Can't Compute False Positive and True Negatice"
    if p+r !=0:
        print "F Square \t:",2*(p*r)/(p+r)
    else:
        print"Can't Compute F Square"
    return p,r
```

In [130]:
```
print "Mean Squared Error" ,mean_squared_error(Y_Test,y_Cap)
for i in np.unique(Y_Test):
    print "\nEvaluation Measures from Confusion Matrix for label",i ,": \n"
```

```
Mean Squared Error 0.438735177866

Evaluation Measures from Confusion Matrix for lablel 0.0 :
Accuracy        : 0.561264822134
Recall          : 0.302083333333
False Negative  : 0.697916666667
Precision       : 0.397260273973
False Positive  : 0.28025477707
True Negative   : 0.71974522293
F Square        : 0.343195266272
(0.3972602739726027, 0.3020833333333333)

Evaluation Measures from Confusion Matrix for lablel 1.0 :
Accuracy        : 0.561264822134
Recall          : 0.71974522293
False Negative  : 0.28025477707
Precision       : 0.627777777778
False Positive  : 0.697916666667
True Negative   : 0.302083333333
F Square        : 0.670623145401
(0.6277777777777778, 0.7197452229299363)
```

We see that performance measures of the classifier is not good as only one feature is considered from a muli feature dataset

nD 2-Class Gaussian Discriminant Analysis

Data Preparation

In [131]:
```
X_multi_0=[]
X_multi_1=[]
for x in X_Train:
    if x[len(x)-1]==0:
        X_multi_0.append(list(x[range(0,len(x)-1)]))
    elif x[len(x)-1]==1:
        X_multi_1.append(list(x[range(0,len(x)-1)]))

X_multi_0 = np.array(X_multi_0)
X_multi_1 = np.array(X_multi_1)
```

Mulitvariate Gaussian Discriminant

$$g_j(X) = -log(|\sum_j|) - 1/2(X-\mu_j)^T\sum^{-1}(X-\mu_j) + log(\alpha_j)$$

In [132]:
```
multi_meanJ,multi_sigmaJ = model_parameters(X_multi_0,X_multi_1,d=2)
multi_y_cap  = predict(X_Test,multi_meanJ,multi_sigmaJ,dim=2 )
```

In [133]:
```
print "Mean Squared Error" ,mean_squared_error(Y_Test,multi_y_cap)
for i in np.unique(Y_Test):
    print "\n Evaluation Measures from Confusion Matrix for label ", i, ": \
    evaluate_performance(multi_y_cap,Y_Test,i)
```

```
Mean Squared Error 0.237154150198

 Evaluation Measures from Confusion Matrix for label  0.0 :

Accuracy        : 0.762845849802
Recall          : 0.552083333333
False Negative  : 0.447916666667
Precision       : 0.757142857143
False Positive  : 0.108280254777
True Negative   : 0.891719745223
F Square        : 0.638554216867

 Evaluation Measures from Confusion Matrix for label  1.0 :

Accuracy        : 0.762845849802
Recall          : 0.891719745223
False Negative  : 0.108280254777
Precision       : 0.765027322404
False Positive  : 0.447916666667
True Negative   : 0.552083333333
F Square        : 0.823529411765
```

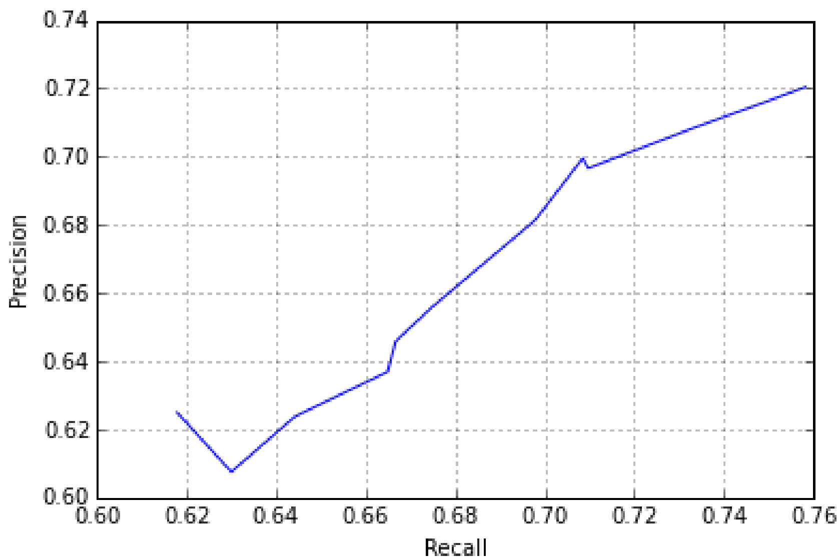
We can see that the performace of the classifier has improved when move features were taken into consideration.

In [134]:
```
def plot_Precision_recall(X_Test,Y_Test,multi_meanJ,multi_sigmaJ,dim=2):
   """Plot the Precision recall curve and find the area under the curve """
    length = X_Test.shape[0]

    perfold = 20

    n=1
    data_pr=[]


    while(n*perfold<=length):
        y_c = predict(X_Test[:n*perfold],multi_meanJ,multi_sigmaJ,dim=2 )
        p=[]
        r=[]
        for i in np.unique(Y_Test):
            a,b,c,d = confussion_matrix(y_c,Y_Test[:n*perfold],i)
            if c+d !=0 and b+d!=0:

                p.append(d/(c+d))
                r.append(d/(b+d))
            else:
                p.append(0)
                p.append(0)


        data_pr.append([np.mean(r),np.mean(p)])
        n+=1

    data_pr= np.array(sorted(data_pr,key=lambda l:l[0]))

    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.subplot(111)

    plt.grid(True)


    plt.plot(data_pr[:,0],data_pr[:,1])

    print "Area Under the PR Curve using Trapezium rule",np.trapz(y=data_pr[
    return data_pr
```

In [135]:
```
plot_Precision_recall(X_Test,Y_Test,multi_meanJ,multi_sigmaJ,dim=2)
```

```
Area Under the PR Curve sing Trapezium rule 7.34445326274
```

Out[135]:
```
array([[ 0.61783961,  0.625     ],
       [ 0.62996032,  0.60755337],
       [ 0.64420063,  0.62380383],
       [ 0.66487455,  0.63690476],
       [ 0.66666667,  0.64583333],
       [ 0.67476489,  0.65583508],
       [ 0.6978022 ,  0.68133224],
       [ 0.70847652,  0.6995614 ],
       [ 0.70968168,  0.69659443],
       [ 0.73386825,  0.70873397],
       [ 0.74825851,  0.7155578 ],
       [ 0.75832698,  0.72048611]])
```



nD k-Class Gaussian Discriminant Analysis

Compute the mean and variance of samples with y=0 and y= 1 separately on X_Train

$\mu_j = 1/m_j \sum_{i=1}^{m_j} I(Y^j = i) X^{(i)}$

$\sum_j = 1/m_j \sum_{i=1}^{m_j} (X^{(i)} - \mu_j)(X^{(i)} - \mu_j)^T$

In [136]:
```
data_nk = getdata_CSV("cleveland.csv")
```

In [137]:
```
reqd_cols = [0,2,3,4,7,9,13]
X_Train = data_nk[:,reqd_cols][:200]
Y_Train = X_Train[:,6]
X_Train = np.delete(X_Train,-1,1)

X_Test = data_nk[:,reqd_cols][200:]

Y_acc = X_Test[:,6]
X_Test = np.delete(X_Test,-1,1)
```

In [138]:
```
X=[]
X= [X_Train[Y_Train==k] for k in np.unique(Y_Train)]
X=np.array(X)
```

In [139]:
```
def kcalss_param(X):
    """ Compute the  K class parameters like Mu and Sigma using the formula
    mean=[]
    sigma=[]
    for i in range(0,len(X)-1):
        mean.append(np.array(np.mean(X[i],axis=0)))

        sigma.append(np.array(np.cov(X[i].T)))

    return np.array(mean) , np.array(sigma)

kmean,ksigma = kcalss_param(X)
```

In [140]:
```
kmulti_y_cap = predict(X_Test,kmean,ksigma,dim=4)
```

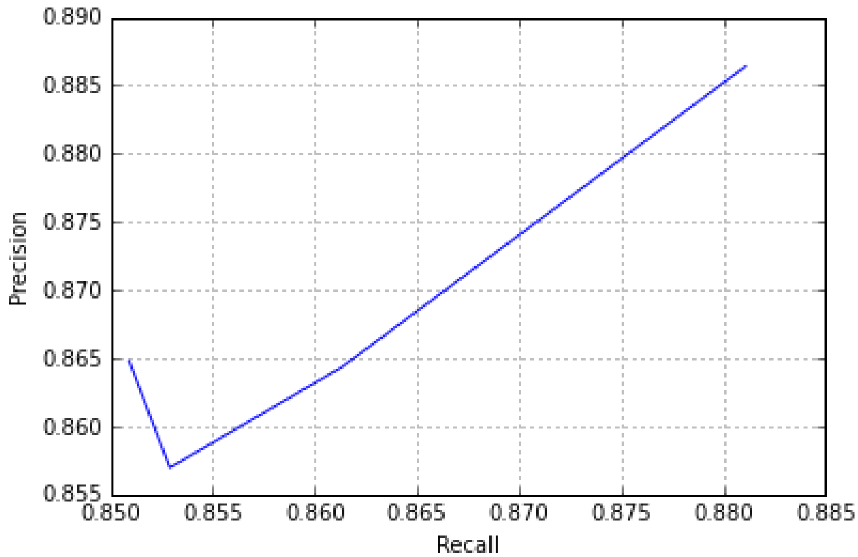
In [141]:
```
print "Mean Squared Error" ,mean_squared_error(Y_acc,kmulti_y_cap)
for i in np.unique(Y_acc):
    print "\n Evaluation Measures from Confusion Matrix for label",i,": \n"
    evaluate_performance(kmulti_y_cap ,Y_acc,i)
```

```
Mean Squared Error 1.36082474227

 Evaluation Measures from Confusion Matrix for label 0.0 :

Accuracy        : 0.701030927835
Recall          : 0.702127659574
False Negative  : 0.297872340426
Precision       : 0.6875
False Positive  : 0.3
True Negative   : 0.7
F Square        : 0.694736842105

 Evaluation Measures from Confusion Matrix for label 1.0 :

Accuracy        : 0.711340206186
Recall          : 0.8125
False Negative  : 0.1875
Precision       : 0.833333333333
False Positive  : 0.764705882353
True Negative   : 0.235294117647
F Square        : 0.822784810127

 Evaluation Measures from Confusion Matrix for label 2.0 :

Accuracy        : 0.752577319588
Recall          : 0.853658536585
False Negative  : 0.146341463415
Precision       : 0.853658536585
False Positive  : 0.8
True Negative   : 0.2
F Square        : 0.853658536585

 Evaluation Measures from Confusion Matrix for label 3.0 :

Accuracy        : 0.835051546392
Recall          : 0.894117647059
False Negative  : 0.105882352941
Precision       : 0.915662650602
False Positive  : 0.583333333333
True Negative   : 0.416666666667
F Square        : 0.904761904762

 Evaluation Measures from Confusion Matrix for label 4.0 :

Accuracy        : 0.969072164948
Recall          : 1.0
False Negative  : 0.0
Precision       : 0.969072164948
False Positive  : 1.0
True Negative   : 0.0
F Square        : 0.984293193717
```

In [142]:
```
plot_Precision_recall(X_Test,Y_acc,kmean,ksigma,dim=4)
```

```
Area Under the PR Curve sing Trapezium rule 2.59695267732
```

Out[142]:
```
array([[ 0.85091503,  0.86480159],
       [ 0.85290563,  0.85697479],
       [ 0.86135215,  0.86437071],
       [ 0.88109388,  0.88641278]])
```



NB - Bernouli

In [17]:
```
def get_data(filename,reqd_cols,train_size=400,test_size=200):
    """Data Preparation , read CSV data set and spli them into Testing and T
    data_nBB = getdata_CSV(filename)

    nBB_X_Train = data_nBB[:,reqd_cols][:train_size]
    nBB_Y_Train = nBB_X_Train[:,len(reqd_cols)-1]
    nBB_X_Train = np.delete(nBB_X_Train,-1,1)

    nBB_X_Test = data_nBB[:,reqd_cols][train_size:train_size+]

    nBB_Y_acc = nBB_X_Test[:,len(reqd_cols)-1]
    nBB_X_Test = np.delete(nBB_X_Test,-1,1)
    return nBB_X_Train,nBB_Y_Train,nBB_X_Test,nBB_Y_acc

reqd_cols = [0,4,8,9,11,12,15]
nBB_X_Train,nBB_Y_Train,nBB_X_Test,nBB_Y_acc = get_data("credit_Score.csv",r
```

In [18]:
```
nBB_X_Train.shape
```

Out[18]:
```
(400L, 6L)
```

Model Parameters

$\alpha_{p|y=1} = 1/m_j \sum_{i=1}^{m_j} X_p^i$

$Prior_i = 1/m_j \sum_{i=1}^{m_j} I(Y^j = i)$

In [161]:
```
def model_params_NB_Bernoulli(nBB_X_Train,nBB_Y_Train):
    alpha_1 =  sum((nBB_X_Train[nBB_Y_Train==1]))/nBB_X_Train[nBB_Y_Train==1
    prior1 = float(nBB_X_Train[nBB_Y_Train==1].shape[0])/len(nBB_X_Train)

    return alpha_1,1-alpha_1,prior1 ,1-prior1
```

In [162]:
```
alpha_1,alpha_2,prior_1,prior_2 = model_params_NB_Bernoulli(nBB_X_Train,nBB_
```

Membership Function using Log Likelihood

$\sum_{i=1}^{n}(X_j log(\alpha_{j/y=i}) + (1 - X_j)log(1 - \alpha_j/y = i)) + log(\alpha_j)$

In [163]:
```
def predict_NB_Bernoulli(X_Test,alpha_1,alpha_2):
    for x in X_Test:
        g0 = sum(x*np.log(alpha_1) + (1-x)*np.log(alpha_2)) + np.log(prior_1
        g1 = sum(x*np.log(alpha_2) + (1-x)*np.log(alpha_1)) + np.log(prior_2

        if g0-g1 > 0:
            y_cap.append(g0)
        else:
            y_cap.append(g1)
    return y_cap

nBB_Y_Cap = predict_NB_Bernoulli(X_Test,alpha_1,alpha_2)
```

In [165]:
```
print "Mean Squared Error" ,mean_squared_error(nBB_Y_acc,y_cap)
for i in np.unique(nBB_Y_acc):
    print "\n Evaluation Measures from Confusion Matrix for label",i,": \n"
    evaluate_performance(y_cap,nBB_Y_acc,i)
```

```
Mean Squared Error 0.399209486166

 Evaluation Measures from Confusion Matrix for label 0.0 :

Accuracy        : 0.600790513834
Recall          : 0.202380952381
False Negative  : 0.797619047619
Precision       : 0.333333333333
False Positive  : 0.201183431953
True Negative   : 0.798816568047
F Square        : 0.251851851852

 Evaluation Measures from Confusion Matrix for label 1.0 :

Accuracy        : 0.600790513834
Recall          : 0.798816568047
False Negative  : 0.201183431953
Precision       : 0.668316831683
False Positive  : 0.797619047619
True Negative   : 0.202380952381
F Square        : 0.727762803235
```

NB- Binomial

In [78]:
```
reqd_cols = range(2,56)
nBBi_X_Train,nBBi_Y_Train,nBBi_X_Test,nBBi_Y_acc = get_data("spambase.csv",r
```

$$\alpha_{j/y=l} = (\sum_{i=1}^{m} I(Y^i = l)X_j^i)/(\sum_{i=1}^{m} I(Y^i = l)P^i)$$

In [79]:
```
def NB_Binomial_params(nBBi_X_Train,nBBi_Y_Train,cls):
    dr=0
    for i in range(0,len(nBBi_Y_Train)-1):
        if(nBBi_Y_Train[i]==cls):
            dr += sum(nBBi_X_Train[i])
    alpha =  sum((nBBi_X_Train[nBBi_Y_Train==cls]))/dr
    return alpha

alpha_0 = NB_Binomial_params(nBBi_X_Train,nBBi_Y_Train,0)
alpha_1 = NB_Binomial_params(nBBi_X_Train,nBBi_Y_Train,1)
```

In [84]:
```
def compute_prior(nBBi_X_Train,nBBi_Y_Train):
    """Computes the prior probabilites for each class """
    prior_0 =  1.*nBBi_X_Train[nBBi_Y_Train==0].shape[0]/nBBi_X_Train.shape[
    prior_1 =  1.*nBBi_X_Train[nBBi_Y_Train==1].shape[0]/nBBi_X_Train.shape[
    return prior_0,prior_1

prior_0,prior_1 = compute_prior(nBBi_X_Train,nBBi_Y_Train)
```

Membership Function

$$g_l(X) = \sum_{j=1}^{n} log(\binom{P}{X_j}\alpha_{j/y=l}^{X_j}(1-\alpha_{j/y=l})^{p-X_j}) + log(\alpha_l)$$

Classification: $\hat{y} = \arg\max_l g_l(X)$

In [85]:
```
#Predict - need to make it as an function
def nBBi_Predict(nBBi_X_Test,alpha_0,alpha_1,prior_0,prior_1):
    y=[]
    for x in nBBi_X_Test:
        g_0=0
        g_1=0
        p=sum(x)

        for i in range(0,len(x)-1):

            g_0 += np.log(math.factorial(p)/math.factorial(x[i])*(alpha_0[i]
            g_1 += np.log(math.factorial(p)/math.factorial(x[i])*(alpha_1[i]
        if g_0-g_1 > 0:
             y.append(0)
        else:
            y.append(1)
    return y


y_cap = nBBi_Predict(nBBi_X_Test,alpha_0,alpha_1,prior_0,prior_1)
```

```
C:\Users\Goutham\Anaconda\lib\site-packages\IPython\kernel\__main__.py:1
2: RuntimeWarning: divide by zero encountered in log
C:\Users\Goutham\Anaconda\lib\site-packages\IPython\kernel\__main__.py:1
1: RuntimeWarning: divide by zero encountered in log
C:\Users\Goutham\Anaconda\lib\site-packages\IPython\kernel\__main__.py:1
3: RuntimeWarning: invalid value encountered in double_scalars
```

In [86]:
```
print "Mean Squared Error" ,mean_squared_error(nBBi_Y_acc,y_cap)
for i in np.unique(nBBi_Y_acc):
    print "\n Evaluation Measures from Confusion Matrix for label",i,": \n"
    evaluate_performance(y_cap,nBBi_Y_acc,i)
```

```
Mean Squared Error 0.15

 Evaluation Measures from Confusion Matrix for label 0.0 :

Accuracy        : 0.85
Recall          : 0.626666666667
False Negative  : 0.373333333333
Precision       : 0.959183673469
False Positive  : 0.016
True Negative   : 0.984
F Square        : 0.758064516129

 Evaluation Measures from Confusion Matrix for label 1.0 :

Accuracy        : 0.85
Recall          : 0.984
False Negative  : 0.016
Precision       : 0.814569536424
False Positive  : 0.373333333333
True Negative   : 0.626666666667
F Square        : 0.891304347826
```

In [ ]:

In [ ]: