In [115]:

```
import numpy as np
import math
#math.log?
data = np.genfromtxt("A4/spam_data.txt",delimiter=None)
#print data
def entropy(y):
    val =0
    total = y.count(-1) + y.count(1)
    p_ofy_1 = y.count(1)/(1.0*total)
    p_ofy_0 = y.count(-1)/(1.0*total)
    #print total
    #print p_ofy_0
    if (p_ofy_1 !=0.0):
        val += (p_ofy_1* math.log(1/p_ofy_1,2))
    else:
        val+=0
    if (p_ofy_0 !=0.0):
        val += (p_ofy_0* math.log(1/p_ofy_0,2))
    else:
        val+=0
    return val

X1 = data[:,0]
X2 = data[:,1]
X3 = data[:,2]
X4 = data[:,3]
X5 = data[:,4]
Y = data[:,5]
y =[]

for i in range(len(Y)):
    y.insert(i,Y[i])

print "entropy of y is %f"%entropy(y)
```

```
entropy of y is 0.970951
```

In [116]:

```
def split(X,Y,yentr):
    x1 =[]
    x0 =[]
    x1count=0
    x0count=0
    for i in range(len(X)):
        if X[i]==1.0:
            x1.insert(i,Y[i])
            x1count+=1
        else:
            x0.insert(i,Y[i])
            x0count+=1
    #print x1count
    #print x1
    #print x0
    return (x1count*(yentr-entropy(x1))+x0count*(yentr -entropy(x0)))/(1.0 * len(X))



y_entr = entropy(y)
#print "extropy of y is %f" %y_entr
#print y_entr
infogain_list=[]
infogain_list.insert(0,split(X1,Y,y_entr))
infogain_list.insert(1,split(X2,Y,y_entr))
infogain_list.insert(2,split(X3,Y,y_entr))
infogain_list.insert(3,split(X4,Y,y_entr))
infogain_list.insert(4,split(X5,Y,y_entr))


print "info gain after first splitting X1 is %f "%infogain_list[0]
print "info gain after first splitting X2 is %f "%infogain_list[1]
print "info gain after first splitting X3 is %f "%infogain_list[2]
print "info gain after first splitting X4 is %f "%infogain_list[3]
print "info gain after first splitting X5 is %f \n"%infogain_list[4]
```

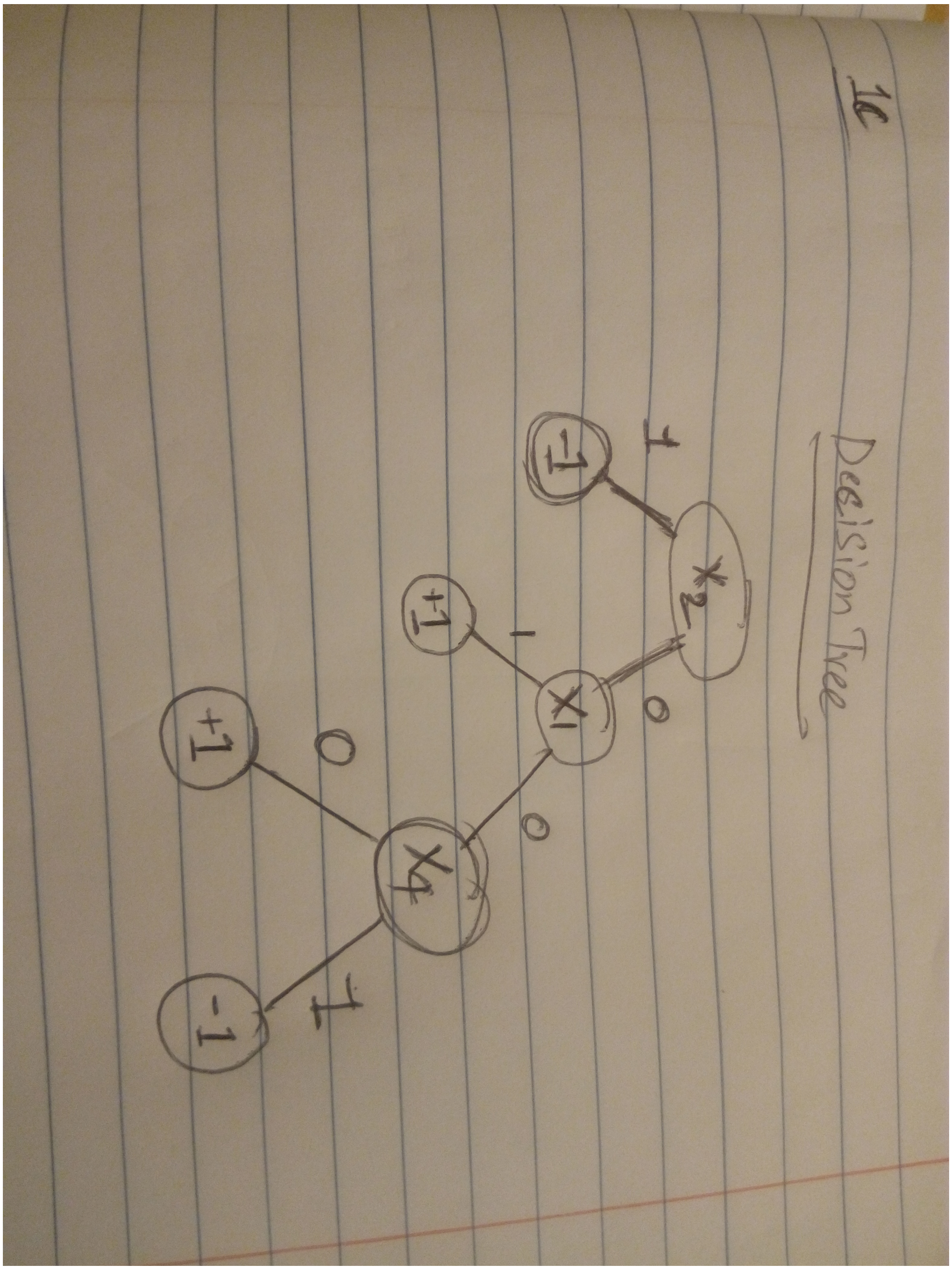
```
info gain after first splitting X1 is 0.046439
info gain after first splitting X2 is 0.609987
info gain after first splitting X3 is 0.005802
info gain after first splitting X4 is 0.091277
info gain after first splitting X5 is 0.005802
```

So the feature to be split first is X2 as the information gain is more in this case.

In [120]:

```
from IPython.display import Image
Image(filename='Dtree.jpg')
```

Out[120]:



In [102]:

```
#2a

import numpy as np
import math
import matplotlib.pyplot as plt
import mltools as ml
import mltools.dtree as dtree
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
#math.log?
#data = np.genfromtxt("A4/spam_data.txt",delimiter=None)
Xtrain=np.genfromtxt("A4/data/X_train.txt",delimiter=None)
Ytrain=np.genfromtxt("A4/data/Y_train.txt",delimiter=None)
#print len(Ytrain)
ytrain=Ytrain[0:10000]
yvalid = Ytrain[10000:20000]
xtrain = Xtrain[0:10000,:]
xvalid = Xtrain[10000:20000,:]
#print len(ytrain)
#print xtrain
```

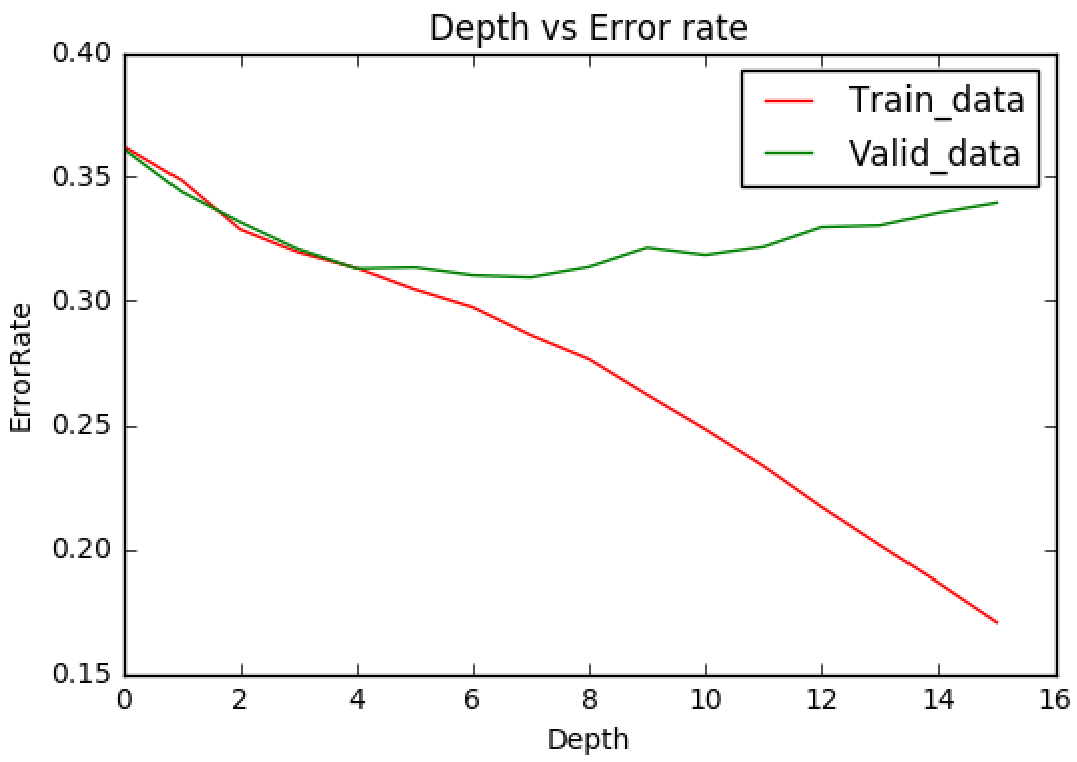
In [103]:

```
#2b

error =0
dt = dtree.treeClassify()

dt.train(xtrain,ytrain, maxDepth=50)
ytr_pred = dt.predict(xtrain)

yval_pred = dt.predict(xvalid)
for i in range(len(ytrain)):
    error += abs((ytrain[i]- ytr_pred[i]))
train_error = error/(len(ytr_pred))
print train_error

error=0
for i in range(len(yvalid)):
    error += abs((yvalid[i]- yval_pred[i]))
valid_error = error/(len(yval_pred))
print valid_error
```

```
0.0125
0.3713
```

In [113]:

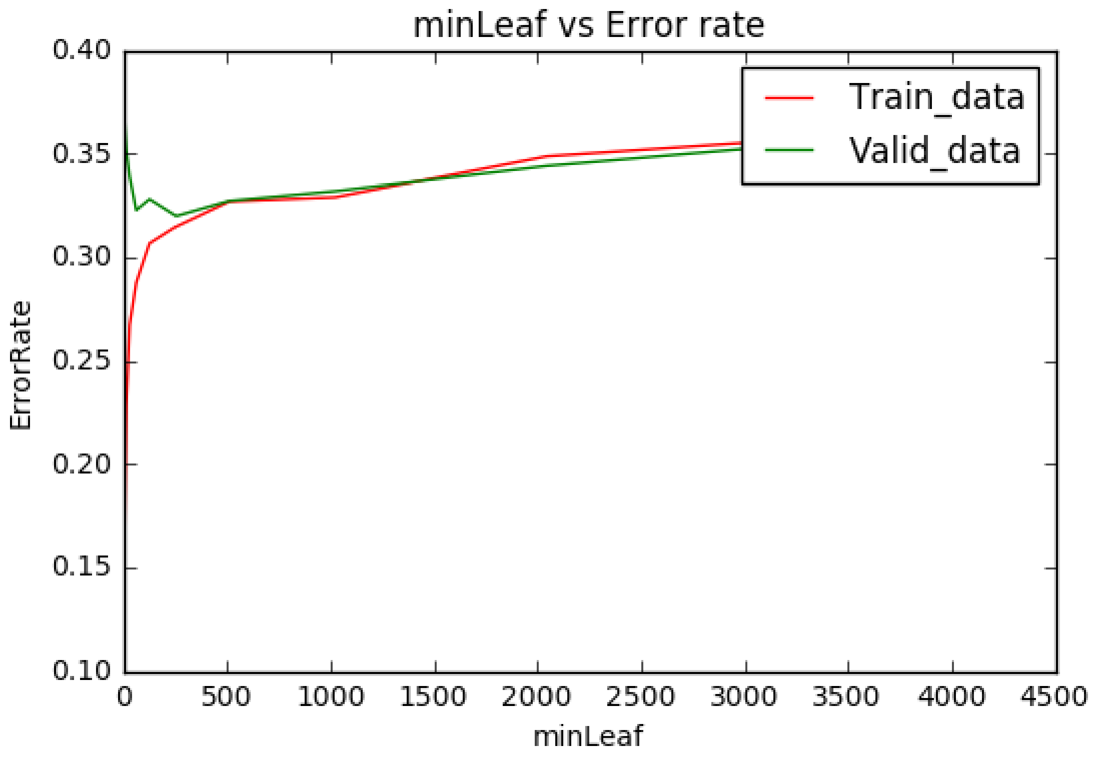
```
#2c

train_error=[]
valid_error=[]
depth=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
for j in range(len(depth)):
    error=0
    dt.train(xtrain,ytrain, maxDepth=depth[j])
    ytr_pred = dt.predict(xtrain)

    yval_pred = dt.predict(xvalid)
    for i in range(len(ytrain)):
        error += abs((ytrain[i]- ytr_pred[i]))
    train_error.insert(j,error/(len(ytr_pred)))
    #print train_error

    error=0
    for i in range(len(yvalid)):
        error += abs((yvalid[i]- yval_pred[i]))
    valid_error.insert(j,error/(len(yval_pred)))
    #print valid_error
plt.clf()
plt.plot(depth,train_error,c='r')
plt.plot(depth,valid_error,c='g')
plt.title("Depth vs Error rate")
plt.xlabel('Depth')
plt.ylabel('ErrorRate')
plt.legend(['Train_data','Valid_data'])
plt.show()

#depth can be 6 or 7
```



From the graph we can see that by increasing depth the complexity starts increasing. The model begins to over fit as more & more branches are created and training error rate decreases from the graph we can see that for depth =4 the validation error is minimum. so the model begins to overfit when depth>4 The best depth would be 4.

In [112]:

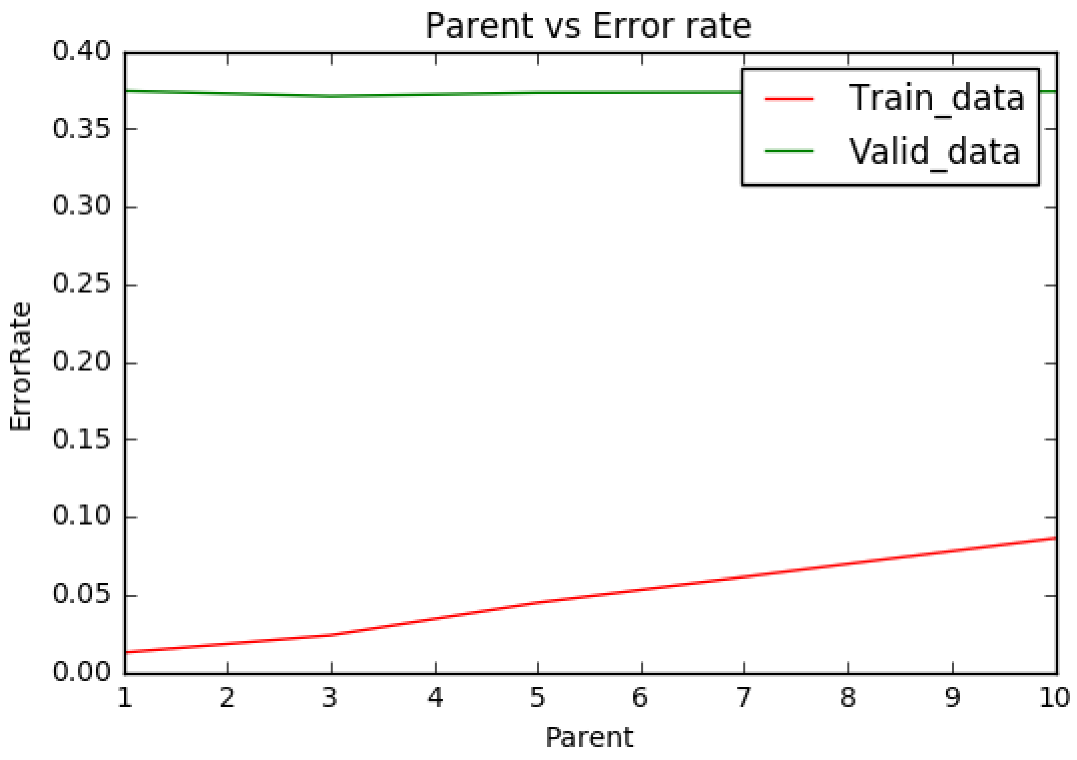
```
#2d

train_error=[]
valid_error=[]

#leaf=2^ list(range(2,13))
leaf=np.power(2,range(2,13))
#print leaf
for j in range(len(leaf)):
    error=0
    dt.train(xtrain,ytrain,minLeaf=leaf[j],maxDepth=50)
    ytr_pred = dt.predict(xtrain)

    yval_pred = dt.predict(xvalid)
    for i in range(len(ytrain)):
        error += abs((ytrain[i]- ytr_pred[i]))
    train_error.insert(j,error/(len(ytr_pred)))
    #print train_error

    error=0
    for i in range(len(yvalid)):
        error += abs((yvalid[i]- yval_pred[i]))
    valid_error.insert(j,error/(len(yval_pred)))
    #print valid_error
plt.clf()
plt.plot(leaf,train_error,c='r')
plt.plot(leaf,valid_error,c='g')
plt.title("minLeaf vs Error rate")
plt.xlabel('minLeaf')
plt.ylabel('ErrorRate')
plt.legend(['Train_data','Valid_data'])
plt.show()
```



From the graph we can see that the training error is not decreasing which means the model is not overfitting. the best value of the minLeaf is 256.

In [114]:

```
#2e

train_error=[]
valid_error=[]
#leaf=2^ list(range(2,13))
parent=[1,3,5,10]
#print leaf
for j in range(len(parent)):
    error=0
    dt.train(xtrain,ytrain,minParent=parent[j],maxDepth=50)
    ytr_pred = dt.predict(xtrain)

    yval_pred = dt.predict(xvalid)
    for i in range(len(ytrain)):
        error += abs((ytrain[i]- ytr_pred[i]))
    train_error.insert(j,error/(len(ytr_pred)))
    #print train_error

    error=0
    for i in range(len(yvalid)):
        error += abs((yvalid[i]- yval_pred[i]))
    valid_error.insert(j,error/(len(yval_pred)))
    #print valid_error
plt.clf()
plt.plot(parent,train_error,c='r')
plt.plot(parent,valid_error,c='g')
plt.title("Parent vs Error rate")
plt.xlabel('Parent')
plt.ylabel('ErrorRate')
plt.legend(['Train_data','Valid_data'])
plt.show()
```

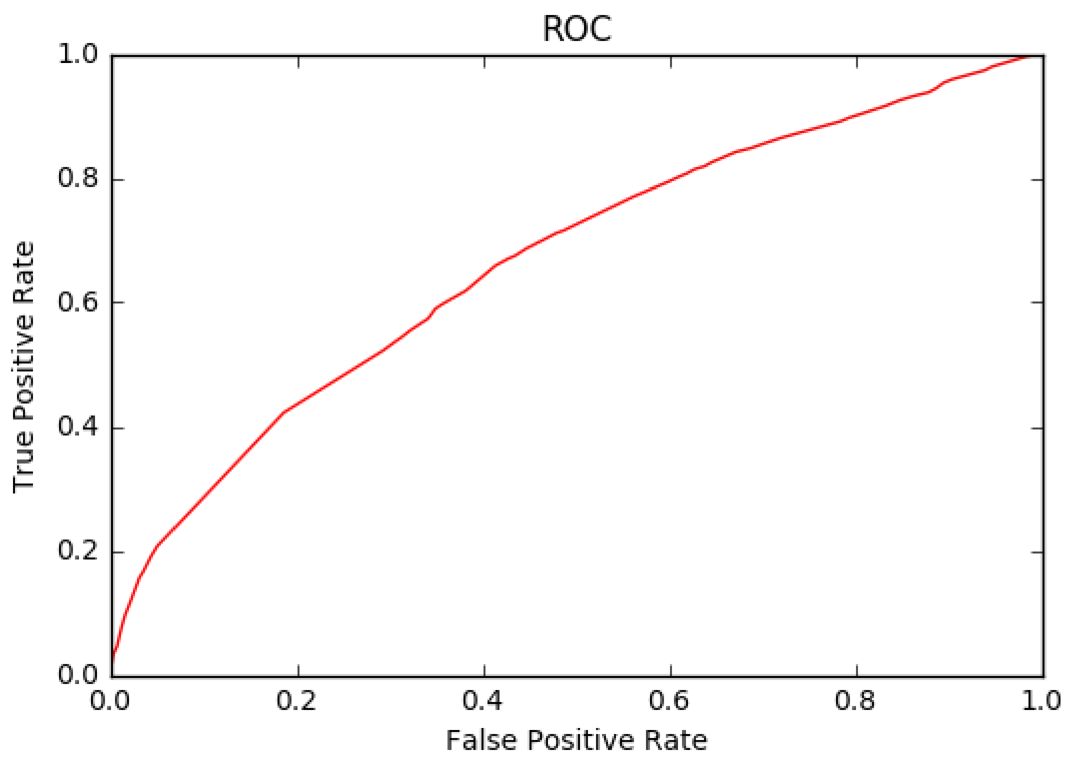


As the min parent increase the complexity decreases.

In [108]:

```
#2f

from sklearn import metrics
ytrain=Ytrain[0:10000]
yvalid = Ytrain[10000:20000]
xtrain = Xtrain[0:10000,:]
xvalid = Xtrain[10000:20000,:]

error=0
dt.train(xtrain,ytrain,minLeaf=64,maxDepth=7)
ytr_pred = dt.predict(xtrain)

yval_pred = dt.predict(xvalid)
for i in range(len(ytrain)):
    error += abs((ytrain[i]- ytr_pred[i]))
train_error=error/(len(ytr_pred))
#print train_error

error=0
for i in range(len(yvalid)):
    error += abs((yvalid[i]- yval_pred[i]))
valid_error= error/(len(yval_pred))
#print valid_error

fpr = []
tpr = []
roc_auc = []

fpr, tpr,threshold  = dt.roc(xvalid,yvalid)
roc_auc = dt.auc(xvalid,yvalid)

print "the area under curve is %f" %roc_auc
plt.clf()
plt.plot(fpr,tpr,c='r')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.show()
```

```
the area under curve is 0.669907
```



In [41]:

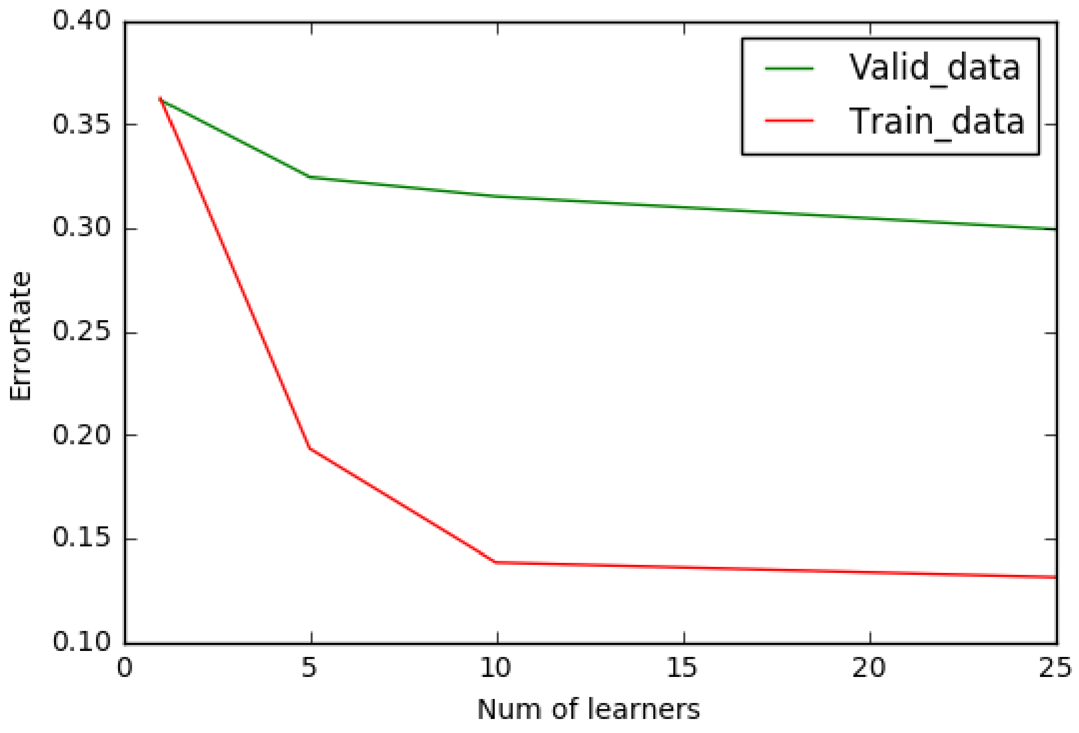
```
#2f

Xtest=np.genfromtxt("A4/data/X_test.txt",delimiter=None)

Ypred = dt.predictSoft(Xtest)
# Now output a file with two columns, a row ID and a confidence in class 1:
np.savetxt('Yhat_knn200.txt',
np.vstack( (np.arange(len(Ypred)) , Ypred[:,1]) ).T,
'%d, %.2f',header='ID,Prob1',comments='',delimiter=',');
```

Uploded the report in kaggle and the performance is 0.63579. this is less than the score calculated from the validation data.

In [110]:

```
#3a

x_bs=[]
y_bs=[]

y_en_train=np.zeros((len(ytrain),25))
y_en_valid=np.zeros((len(yvalid),25))
ensemble=[]
valid_error=[]
train_error=[]


#print len(ytrain)

for i in range(0,25):
    x_bs,y_bs= ml.bootstrapData(xtrain,ytrain,len(ytrain))
    ensemble.insert(i,dtree.treeClassify())
    ensemble[i].train(x_bs,y_bs,nFeatures=4,minLeaf=4,maxDepth=20)
    y_en_valid[:,i]= ensemble[i].predict(xvalid)
    y_en_train[:,i] = ensemble[i].predict(xtrain)

for i,m in enumerate([1,5,10,25]):
    y_valid_final=[]
    y_train_final=[]
    #valid_error_en=0
    #train_error_en=0
    y_valid_final = (np.mean(y_en_valid[:,0:m-1],axis=1) > 0.5)
    y_train_final = (np.mean(y_en_train[:,0:m-1],axis=1) > 0.5)
    valid_error_en = np.mean(y_valid_final!=yvalid)
    valid_error.insert(i,valid_error_en)
    train_error_en = np.mean((y_train_final!=ytrain))
    train_error.insert(i,train_error_en)
print train_error
print valid_error

plt.clf()
plt.plot([1,5,10,25],valid_error,'g')
plt.plot([1,5,10,25],train_error,'r')
plt.xlabel("Num of learners")
plt.ylabel("ErrorRate")
plt.legend(['Valid_data','Train_data'])
plt.show()
```

```
[0.36249999999999999, 0.1933, 0.1384, 0.1313]
[0.36149999999999999, 0.32440000000000002, 0.31519999999999998, 0.29930000
000000001]
```



In [96]:

```
size =25
#print Xtest.shape
y_en_valid = np.zeros((len(Xtest),25))
y_valid_final=np.zeros((len(Xtest)))
#y_en_valid=[None]*int(Xtest.shape[0],25)
for i in range(0,size):
    y_en_valid[:,i]= ensemble[i].predictSoft(Xtest)[:,1]

y_valid_final = (np.mean(y_en_valid[:,0:24],axis=1))
#print len(y_valid_final)
#print y_valid_final
# Now output a file with two columns, a row ID and a confidence in class 1:
np.savetxt('Yhat_knn200.txt',
np.vstack((np.arange(len(y_valid_final)) , y_valid_final)).T,
'%d, %.2f',header='ID,Prob1',comments='',delimiter=',');
```

Ensemble size chosen is 25, Uploaded the report in kaggle and the score obtained is 0.63240