```
1a. H(y) = (0.6 * log2(5/3)) + (0.4 * log2(5/2)) = 0.971
```

```
1b.
    H(x1) = (0.6 * (0.971 - (0.5 * log2(2) + 0.5 * log2(2)))) + (0.4 *
(0.971 - (0.75 * log2(4/3) + 0.25 * log2(4)) = 0.6 + (0.4 * 0.811) =
0.06388875022 - 0.0174 = 0.04648875022

    H(x2) = (0.5 * (0.971 - (0.2 * log2(5) + 0.8 * log2(5/4)))) + (0.5 *
(0.971 - (1 * log2(1)))) = (0.5 * 0.722) + (0) =  0.1245359526 + 0.4855 =
0.6100359526

    H(x3) = (0.3 * (0.971 - (0.33 * log2(3) + 0.66 * log2(3/2)))) + (0.7 *
(0.971 - (4/7 * log2(7/4) + 3/7 * log2(7/3))) = 0.01856613729 +
-0.009959695224 = 0.008606442066

    H(x4) = (0.3 * (0.971 - (0.33 * log2(3) + 0.66 * log2(3/2)))) + (0.7 *
(0.971 - (5/7 * log2(7/5) + 2/7 * log2(7/2))) = 0.01856613729 +
0.075515602 = 0.09408173929

        H(x5) = (0.7 * (0.971 - (4/7 * log2(7/4) + 3/7 * log2(7/3)))) +
(0.3 * (0.97 - (0.33 * log2(3) + 0.66 * log2(3/2)))) = -0.009959695224 +
0.01856613729 = 0.008606442066

    Splitting on x2 gives the highest infomation gain.
```

```
2a.
```

In [66]:

```
import numpy as np
import mltools as ml
np.random.seed(0)
X = np.genfromtxt('data/X_train.txt', delimiter=',')
Y = np.genfromtxt('data/Y_train.txt', delimiter=',')
X,Y = ml.shuffleData(X,Y)
X = X[:,:41]
print("Minimum of first feature:", min(X[:,0]))
print("Maximum of first feature:", max(X[:,0]))
print("Mean of first feature:", np.mean(X[:,0]))
print("Variance of first feature:", np.var(X[:,0]))

print()
print("Minimum of second feature:", min(X[:,1]))
print("Maximum of second feature:", max(X[:,1]))
print("Mean of second feature:", np.mean(X[:,1]))
print("Variance of second feature:", np.var(X[:,1]))

print()
print("Minimum of third feature:", min(X[:,2]))
print("Maximum of third feature:", max(X[:,2]))
print("Mean of third feature:", np.mean(X[:,2]))
print("Variance of third feature:", np.var(X[:,2]))

print()
print("Minimum of four feature:", min(X[:,3]))
print("Maximum of four feature:", max(X[:,3]))
print("Mean of four feature:", np.mean(X[:,3]))
print("Variance of four feature:", np.var(X[:,3]))

print()
print("Minimum of fifth feature:", min(X[:,4]))
print("Maximum of fifth feature:", max(X[:,4]))
print("Mean of fifth feature:", np.mean(X[:,4]))
print("Variance of fifth feature:", np.var(X[:,4]))
#Xva =
#learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=50)
```

```
Minimum of first feature: 0.0
Maximum of first feature: 110285.0
Mean of first feature: 1321.1174134446987
Variance of first feature: 6747189.595085322

Minimum of second feature: 0.0
Maximum of second feature: 35.0
Mean of second feature: 6.5916745251246125
Variance of second feature: 34.70690630279573

Minimum of third feature: 0.0
Maximum of third feature: 51536.0
Mean of third feature: 1152.273237235619
Variance of third feature: 5376518.288798102

Minimum of four feature: 0.0
Maximum of four feature: 21768.0
Mean of four feature: 234.8262548834703
```

```
Variance of four feature: 260120.83053297663

Minimum of fifth feature: 0.0
Maximum of fifth feature: 27210.0
Mean of fifth feature: 289.75871211100633
Variance of fifth feature: 406615.8651128233
```

2b.

In [67]:

```python
#Xtr = X[:(int(len(X)/2)),:]
#Ytr = Y[:int(len(Y)/2)]
#Xva = X[(int(len(X)/2)):,:]
#Yva = Y[int(len(Y)/2):]
Xtr,Xva,Ytr,Yva = ml.splitData(X,Y,0.5)
learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=50)
Yhat = learner.predict(Xtr)
Yhat2 = learner.predict(Xva)
print()
print("Training Error:", np.sum(Yhat != Ytr)/len(Ytr))
print("Validation Error:", np.sum(Yhat2 != Yva)/len(Yva))
```

```

Training Error: 0.0
Validation Error: 0.40878469415251956
```

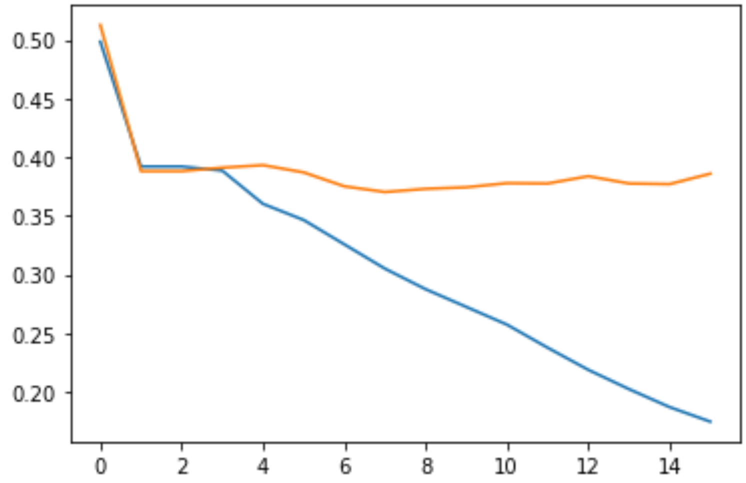
2c.

In [68]:
```
import matplotlib.pyplot as plt
trainerr = []
validerr = []
maxdepth = []
for i in range(16):
    maxdepth.append(i)
    learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=i)
    Yhat = learner.predict(Xtr)
    Yhat2 = learner.predict(Xva)
    trainerr.append(np.sum(Yhat != Ytr)/len(Ytr))
    validerr.append(np.sum(Yhat2 != Yva)/len(Yva))
    print()
    print("Training Error for max depth", i, ":", np.sum(Yhat != Ytr)/len(Y
    print("Validation Error for max depth", i, ":", np.sum(Yhat2 != Yva)/le
plt.plot(maxdepth,trainerr)
plt.plot(maxdepth,validerr)
```

```

Training Error for max depth 0 : 0.4983836206896552
Validation Error for max depth 0 : 0.5125303152789006

Training Error for max depth 1 : 0.39197198275862066
Validation Error for max depth 1 : 0.3880355699272433

Training Error for max depth 2 : 0.39197198275862066
Validation Error for max depth 2 : 0.3880355699272433

Training Error for max depth 3 : 0.3884698275862069
Validation Error for max depth 3 : 0.39099973053085424

Training Error for max depth 4 : 0.36018318965517243
Validation Error for max depth 4 : 0.39315548369711667

Training Error for max depth 5 : 0.3464439655172414
Validation Error for max depth 5 : 0.3869576933441121

Training Error for max depth 6 : 0.32570043103448276
Validation Error for max depth 6 : 0.37510105092966856

Training Error for max depth 7 : 0.30495689655172414
Validation Error for max depth 7 : 0.370250606305578

Training Error for max depth 8 : 0.2874461206896552
Validation Error for max depth 8 : 0.37294529776340607

Training Error for max depth 9 : 0.2723599137931034
Validation Error for max depth 9 : 0.3742926434923201

Training Error for max depth 10 : 0.2572737068965517
Validation Error for max depth 10 : 0.37779574238749664

Training Error for max depth 11 : 0.23760775862068967
Validation Error for max depth 11 : 0.37752627324171384

Training Error for max depth 12 : 0.21875
Validation Error for max depth 12 : 0.3837240635947184
```

```
Training Error for max depth 13 : 0.2023168103448276
Validation Error for max depth 13 : 0.37752627324171384

Training Error for max depth 14 : 0.1869612068965517
Validation Error for max depth 14 : 0.3769873349501482

Training Error for max depth 15 : 0.17456896551724138
Validation Error for max depth 15 : 0.3858798167609809
```

```
Out[68]: [<matplotlib.lines.Line2D at 0x120127a60>]
```



Models with higher max depth have higher complexity. Max depth 7 provides the best model.

2d.

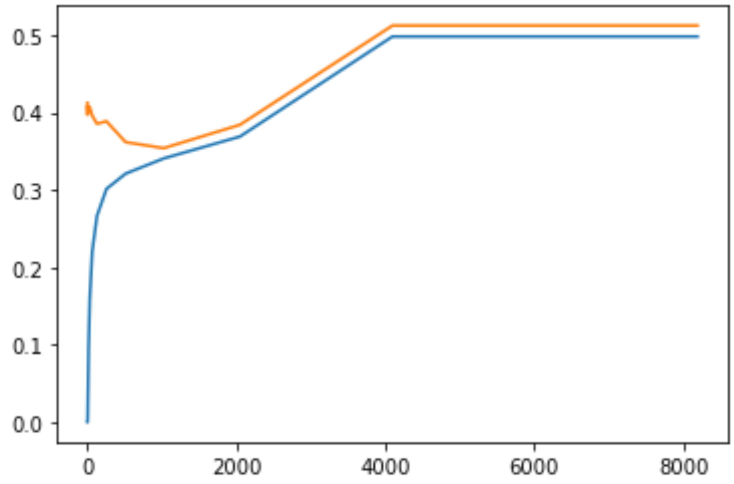
In [69]:

```
import matplotlib.pyplot as plt
trainerr = []
validerr = []
maxdepth = []
for i in range(14):
    x = pow(2,i)
    maxdepth.append(x)
    learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=50, minParent = x)
    Yhat = learner.predict(Xtr)
    Yhat2 = learner.predict(Xva)
    trainerr.append(np.sum(Yhat != Ytr)/len(Ytr))
    validerr.append(np.sum(Yhat2 != Yva)/len(Yva))
    print()
    print("Training Error for min parent", x, ":", np.sum(Yhat != Ytr)/len(
    print("Validation Error for min parent", x, ":", np.sum(Yhat2 != Yva)/l
plt.plot(maxdepth,trainerr)
plt.plot(maxdepth,validerr)
```

```

Training Error for min parent 1 : 0.0
Validation Error for min parent 1 : 0.4125572621934788

Training Error for min parent 2 : 0.0
Validation Error for min parent 2 : 0.3971975208838588

Training Error for min parent 4 : 0.009428879310344827
Validation Error for min parent 4 : 0.40905416329830235

Training Error for min parent 8 : 0.03879310344827586
Validation Error for min parent 8 : 0.4033953112368634

Training Error for min parent 16 : 0.09617456896551724
Validation Error for min parent 16 : 0.4074373484236055

Training Error for min parent 32 : 0.15975215517241378
Validation Error for min parent 32 : 0.40662894098625707

Training Error for min parent 64 : 0.2200969827586207
Validation Error for min parent 64 : 0.39611964430072755

Training Error for min parent 128 : 0.26643318965517243
Validation Error for min parent 128 : 0.3856103476151981

Training Error for min parent 256 : 0.30145474137931033
Validation Error for min parent 256 : 0.38884397736459175

Training Error for min parent 512 : 0.32112068965517243
Validation Error for min parent 512 : 0.36189706278631095

Training Error for min parent 1024 : 0.3407866379310345
Validation Error for min parent 1024 : 0.35408245755860956

Training Error for min parent 2048 : 0.36907327586206895
Validation Error for min parent 2048 : 0.38426300188628404

Training Error for min parent 4096 : 0.4983836206896552
Validation Error for min parent 4096 : 0.5125303152789006
```

```
Training Error for min parent 8192 : 0.4983836206896552
Validation Error for min parent 8192 : 0.5125303152789006
```

```
Out[69]: [<matplotlib.lines.Line2D at 0x122006340>]
```

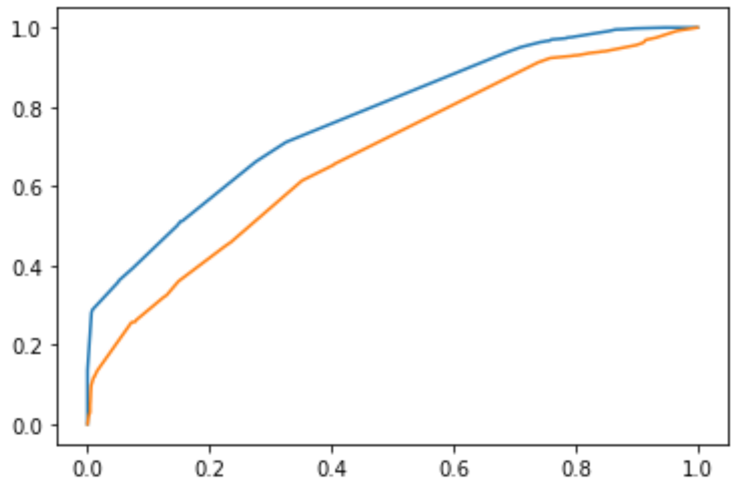


Models with higher minParent have lower complexity. MinParent 1024 gives the best decision tree model.

2f.

In [70]:
```
from sklearn.metrics import roc_curve, auc
import mltools as ml

learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=7)

print("Max depth 7 ROC curve")

fpr, tpr, tnr = learner.roc(Xtr,Ytr)
plt.plot(fpr, tpr)

fpr, tpr, tnr = learner.roc(Xva,Yva)
plt.plot(fpr, tpr)

print("Max depth 7 training error AUC:", learner.auc(Xtr,Ytr))
print("Max depth 7 validation error AUC:", learner.auc(Xva,Yva))
print()
print("Max depth 7 ROC curves for training and validation error:")
```

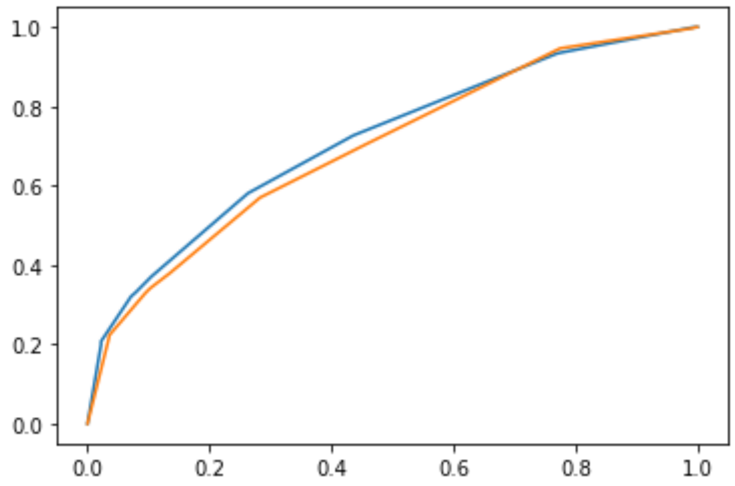
```
Max depth 5 ROC curve
Max depth 7 training error AUC: 0.7754431445408889
Max depth 7 validation error AUC: 0.6790255696630761

Max depth 7 ROC curves for training and validation error:
```



In [71]:

```
from sklearn.metrics import roc_curve, auc
import mltools as ml

learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=50, minParent = 1024)

fpr, tpr, tnr = learner.roc(Xtr,Ytr)
plt.plot(fpr, tpr)

fpr, tpr, tnr = learner.roc(Xva,Yva)
plt.plot(fpr, tpr)
#roc_curve(Yhat, Ytr)

print("Max depth 50 and min parent 1024 training error AUC:", learner.auc(X
print("Max depth 50 and min parent 1024 validation error AUC:", learner.auc
print()
print("Max depth 50 and min parent 1024 ROC curves for training and validat
```

```
Max depth 50 and min parent 1024 training error AUC: 0.7184301971144076
Max depth 50 and min parent 1024 validation error AUC: 0.69812637943592

Max depth 50 and min parent 1024 ROC curves for training and validation e
rror:
```



```
2g. Kaggle username: Joshua Hsin, AUC: 0.72468
```

In [129]:

```
import numpy as np
import mltools as ml
#X = X[:,:41]

learner = ml.dtree.treeClassify(X, Y, maxDepth=7, minParent = 300)
Xte = np.genfromtxt('data/X_test.txt', delimiter=',')
Yte = np.vstack((np.arange(Xte.shape[0]), learner.predictSoft(Xte)[:,1])).T

# Output a file with two columns, a row ID and a confidence in class 1:
np.savetxt('Y_submit.txt',Yte,'%d, %.2f',header='Id,Predicted',comments='',
```

In [117]:

```
X = np.genfromtxt('data/X_train.txt', delimiter=',')
Y = np.genfromtxt('data/Y_train.txt', delimiter=',')
X,Y = ml.shuffleData(X,Y)
X = X[:,:41]

learner = ml.dtree.treeClassify(X, Y, maxDepth = 7, minParent = 300)

print("AUC:", learner.auc(X,Y))
Yhat = learner.predict(X)
print("Error for optimal classifier:", np.sum(Yhat != Y)/len(Y))
```

```
AUC: 0.7329601094616423
Error for optimal classifier: 0.3443351744577664
```

```
3a. Kaggle username: Joshua Hsin, AUC: 0.70632
```

In [118]:

```
values = [0] * 8
index = [1,5,10,20,25,30,40,50]
values2 = [0] * 8
import random
import numpy as np
#np.random.seed(4)
m,n = Xtr.shape
classifiers = [None] * 50
for i in range(50):
    X,Y = ml.bootstrapData(Xtr, Ytr, int(len(Xtr)/50))
    ind = random.randint(1,len(Xtr))
    classifiers[i] = ml.dtree.treeClassify(X, Y, maxDepth = 30, minLeaf = 4

mValid = Xva.shape[0]
predict = np.zeros((mValid, 50))
for i in range(50):
    predict[:,i] = classifiers[i].predict(Xva)
predict2 = np.mean(predict,axis = 1)>0.5

print("Validation Error:", np.sum(predict2 != Yva)/len(Yva))
values[7] = np.sum(predict2 != Yva)/len(Yva)

mTrain = Xtr.shape[0]
predict = np.zeros((mTrain, 50))
for i in range(50):
    predict[:,i] = classifiers[i].predict(Xtr)
predict2 = np.mean(predict,axis = 1)>0.5

print("Training Error:", np.sum(predict2 != Ytr)/len(Ytr))
values2[7] = np.sum(predict2 != Ytr)/len(Ytr)
```

```
Validation Error: 0.39153866882241983
Training Error: 0.3661099137931034
```

In [119]:

```
import random
import numpy as np
#np.random.seed(4)
m,n = Xtr.shape
classifiers = [None] * 40
for i in range(40):
    X,Y = ml.bootstrapData(Xtr, Ytr, int(len(Xtr)/40))
    ind = random.randint(1,len(Xtr))
    classifiers[i] = ml.dtree.treeClassify(X, Y, maxDepth = 30, minLeaf = 4

mValid = Xva.shape[0]
predict = np.zeros((mValid, 40))
for i in range(40):
    predict[:,i] = classifiers[i].predict(Xva)
predict2 = np.mean(predict,axis = 1)>0.5

print("Validation Error:", np.sum(predict2 != Yva)/len(Yva))

values[6] = np.sum(predict2 != Yva)/len(Yva)

mTrain = Xtr.shape[0]
predict = np.zeros((mTrain, 40))
for i in range(40):
    predict[:,i] = classifiers[i].predict(Xtr)
predict2 = np.mean(predict,axis = 1)>0.5

print("Training Error:", np.sum(predict2 != Ytr)/len(Ytr))
values2[6] = np.sum(predict2 != Ytr)/len(Ytr)
```

```
Validation Error: 0.3947722985718135
Training Error: 0.35614224137931033
```

In [120]:

```
import random
import numpy as np
#np.random.seed(4)
m,n = Xtr.shape
classifiers = [None] * 30
for i in range(30):
    X,Y = ml.bootstrapData(Xtr, Ytr, int(len(Xtr)/30))
    ind = random.randint(1,len(Xtr))
    classifiers[i] = ml.dtree.treeClassify(X, Y, maxDepth = 30, minLeaf = 4

mValid = Xva.shape[0]
predict = np.zeros((mValid, 30))
for i in range(30):
    predict[:,i] = classifiers[i].predict(Xva)
predict2 = np.mean(predict,axis = 1)>0.5

print("Validation Error:", np.sum(predict2 != Yva)/len(Yva))

values[5] = np.sum(predict2 != Yva)/len(Yva)

mTrain = Xtr.shape[0]
predict = np.zeros((mTrain, 30))
for i in range(30):
    predict[:,i] = classifiers[i].predict(Xtr)
predict2 = np.mean(predict,axis = 1)>0.5

print("Training Error:", np.sum(predict2 != Ytr)/len(Ytr))
values2[5] = np.sum(predict2 != Ytr)/len(Ytr)
```

```
Validation Error: 0.40231743465373215
Training Error: 0.3415948275862069
```

In [121]:

```
import random
import numpy as np
#np.random.seed(4)
m,n = Xtr.shape
classifiers = [None] * 25
for i in range(25):
    X,Y = ml.bootstrapData(Xtr, Ytr, int(len(Xtr)/25))
    ind = random.randint(1,len(Xtr))
    classifiers[i] = ml.dtree.treeClassify(X, Y, maxDepth = 30, minLeaf = 4

mValid = Xva.shape[0]
predict = np.zeros((mValid, 25))
for i in range(25):
    predict[:,i] = classifiers[i].predict(Xva)
predict2 = np.mean(predict,axis = 1)>0.5

print("Validation Error:", np.sum(predict2 != Yva)/len(Yva))

values[4] = np.sum(predict2 != Yva)/len(Yva)

mTrain = Xtr.shape[0]
predict = np.zeros((mTrain, 25))
for i in range(25):
    predict[:,i] = classifiers[i].predict(Xtr)
predict2 = np.mean(predict,axis = 1)>0.5

print("Training Error:", np.sum(predict2 != Ytr)/len(Ytr))
values2[4] = np.sum(predict2 != Ytr)/len(Ytr)
```

```
Validation Error: 0.375909458367017
Training Error: 0.34509698275862066
```

In [122]:

```
import random
import numpy as np
#np.random.seed(4)
m,n = Xtr.shape
classifiers = [None] * 20
for i in range(20):
    X,Y = ml.bootstrapData(Xtr, Ytr, int(len(Xtr)/20))
    ind = random.randint(1,len(Xtr))
    classifiers[i] = ml.dtree.treeClassify(X, Y, maxDepth = 30, minLeaf = 4

mValid = Xva.shape[0]
predict = np.zeros((mValid, 20))
for i in range(20):
    predict[:,i] = classifiers[i].predict(Xva)
predict2 = np.mean(predict,axis = 1)>0.5

print("Validation Error:", np.sum(predict2 != Yva)/len(Yva))

values[3] = np.sum(predict2 != Yva)/len(Yva)

mTrain = Xtr.shape[0]
predict = np.zeros((mTrain, 20))
for i in range(20):
    predict[:,i] = classifiers[i].predict(Xtr)
predict2 = np.mean(predict,axis = 1)>0.5

print("Training Error:", np.sum(predict2 != Ytr)/len(Ytr))
values2[3] = np.sum(predict2 != Ytr)/len(Ytr)
```

```
Validation Error: 0.3990838049043385
Training Error: 0.34509698275862066
```

In [123]:

```
import random
import numpy as np
m,n = Xtr.shape
classifiers = [None] * 10
for i in range(10):
    X,Y = ml.bootstrapData(Xtr, Ytr, int(len(Xtr)/10))
    ind = random.randint(1,len(Xtr))
    classifiers[i] = ml.dtree.treeClassify(X, Y, maxDepth = 30, minLeaf = 4

mValid = Xva.shape[0]
predict = np.zeros((mValid, 10))
for i in range(10):
    predict[:,i] = classifiers[i].predict(Xva)
predict2 = np.mean(predict,axis = 1)>0.5

print("Validation Error:", np.sum(predict2 != Yva)/len(Yva))

values[2] = np.sum(predict2 != Yva)/len(Yva)

mTrain = Xtr.shape[0]
predict = np.zeros((mTrain, 10))
for i in range(10):
    predict[:,i] = classifiers[i].predict(Xtr)
predict2 = np.mean(predict,axis = 1)>0.5

print("Training Error:", np.sum(predict2 != Ytr)/len(Ytr))
values2[2] = np.sum(predict2 != Ytr)/len(Ytr)
```

```
Validation Error: 0.38938291565615735
Training Error: 0.33836206896551724
```

In [124]:

```
import random
import numpy as np
m,n = Xtr.shape
classifiers = [None] * 5
for i in range(5):
    X,Y = ml.bootstrapData(Xtr, Ytr, int(len(Xtr)/5))
    ind = random.randint(1,len(Xtr))
    classifiers[i] = ml.dtree.treeClassify(X, Y, maxDepth = 30, minLeaf = 4

mValid = Xva.shape[0]
predict = np.zeros((mValid, 5))
for i in range(5):
    predict[:,i] = classifiers[i].predict(Xva)
predict2 = np.mean(predict,axis = 1)>0.5

print("Validation Error:", np.sum(predict2 != Yva)/len(Yva))

values[1] = np.sum(predict2 != Yva)/len(Yva)

mTrain = Xtr.shape[0]
predict = np.zeros((mTrain, 5))
for i in range(5):
    predict[:,i] = classifiers[i].predict(Xtr)
predict2 = np.mean(predict,axis = 1)>0.5

print("Training Error:", np.sum(predict2 != Ytr)/len(Ytr))
values2[1] = np.sum(predict2 != Ytr)/len(Ytr)
```

```
Validation Error: 0.3858798167609809
Training Error: 0.28502155172413796
```

In [125]:

```
import random
import numpy as np
m,n = Xtr.shape
classifiers = [None] * 1
for i in range(1):
    X,Y = ml.bootstrapData(Xtr, Ytr, int(len(Xtr)))
    ind = random.randint(1,len(Xtr))
    classifiers[i] = ml.dtree.treeClassify(X, Y, maxDepth = 30, minLeaf = 4

mValid = Xva.shape[0]
predict = np.zeros((mValid, 1))
for i in range(1):
    predict[:,i] = classifiers[i].predict(Xva)
predict2 = np.mean(predict,axis = 1)>0.5

print("Validation Error:", np.sum(predict2 != Yva)/len(Yva))

values[0] = np.sum(predict2 != Yva)/len(Yva)

mTrain = Xtr.shape[0]
predict = np.zeros((mTrain, 1))
for i in range(1):
    predict[:,i] = classifiers[i].predict(Xtr)
predict2 = np.mean(predict,axis = 1)>0.5

print("Training Error:", np.sum(predict2 != Ytr)/len(Ytr))
values2[0] = np.sum(predict2 != Ytr)/len(Ytr)
```

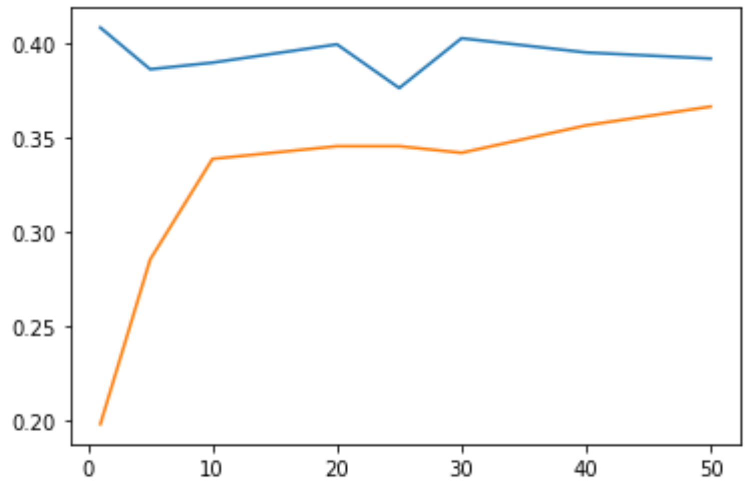
```
Validation Error: 0.4079762867151711
Training Error: 0.19773706896551724
```

In [126]:

```
import matplotlib.pyplot as plt
plt.plot(index, values)
plt.plot(index, values2)
```

Out[126]:

```
[<matplotlib.lines.Line2D at 0x122419760>]
```



In [ ]:

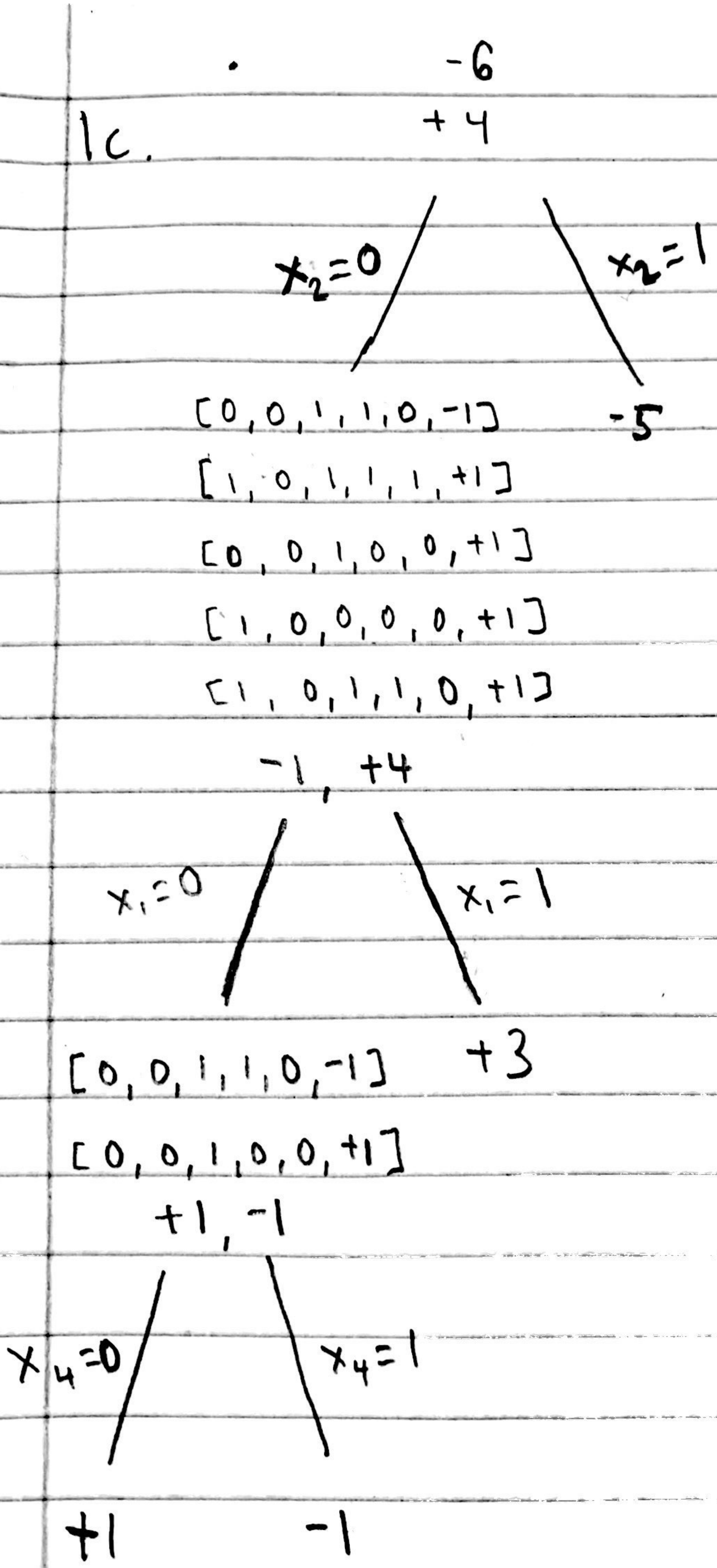
```
3a.
```

In [132]:

```
import random
import numpy as np
Xte = np.genfromtxt('data/X_test.txt', delimiter=',')
X = np.genfromtxt('data/X_train.txt', delimiter=',')
Y = np.genfromtxt('data/Y_train.txt', delimiter=',')
X,Y = ml.shuffleData(X,Y)
X = X[:,:41]
#np.random.seed(4)
m,n = X.shape
classifiers = [None] * 25
for i in range(25):
    Xa,Ya = ml.bootstrapData(X, Y, int(len(X)/25))
    ind = random.randint(1,len(X))
    classifiers[i] = ml.dtree.treeClassify(Xa, Ya, maxDepth = 30,minLeaf =

mValid = Xte.shape[0]
predict = np.zeros((mValid, 25))
for i in range(25):
    predict[:,i] = classifiers[i].predict(Xte)
predict2 = np.mean(predict,axis = 1)

Yte = np.vstack((np.arange(Xte.shape[0]), predict2)).T

np.savetxt('Y_submit.txt',Yte,'%d, %.2f',header='Id,Predicted',comments='',
```

Statement of Collaboration: Discussed the difference between entropy and information gain in the context of problem 1.2, discussed the technique for problem 1.3, and discussed logistics for calculating roc and auc in 2f.

1c.


-6
+4
$x_2=0$
$x_2=1$
[0, 0, 1, 1, 0, -1]
[1, 0, 1, 1, 1, +1]
[0, 0, 1, 0, 0, +1]
[1, 0, 0, 0, 0, +1]
[1, 0, 1, 1, 0, +1]
-5
-1, +4
$x_1=0$
$x_1=1$
[0, 0, 1, 1, 0, -1]
[0, 0, 1, 0, 0, +1]
+3
+1, -1
$x_4=0$
$x_4=1$
+1
-1