Problem 1:

1.1:

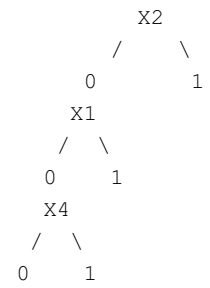
Entropy: H(y) = -(6/10log2(6/10) + 4/10log2(4/10)) = 0.97095059

1.2:

```
X1: 0.97095059 - (6/10(1/2log2(2) + 1/2log2(2)) + 4/10(3/4log2(4/3) + 1/4log2(4)))
    = 0.04643934
X2: 0.97095059 - (1/2(0log2(0) + 5log2(5)) + 1/2(4/5log2(5/4) + 1/5log2(5)))
    = 0.60998654            #Log2(0) is undefined so I considered the left side as 0.
X3: 0.97095059 - (7/10(3/7log2(7/3) + 4/7log2(7/4)) + 3/10(1/3log2(3) + 2/3log2(3/2)))
    = 0.00580214
X4: 0.97095059 - (7/10(2/7log2(7/2) + 5/7log2(7/5)) + 3/10(2/3log2(3/2) + 1/4log2(4)))
    = 0.09127744
X5: 0.97095059 - (3/10(1/3log2(3) + 2/3log2(3/2)) + 7/10(3/7log2(7/3) + 4/7log2(7/4)))
    = 0.00580214

#Calculations computed on a calculator

You would split the root node at the feature with the highest information gain, which is
X2.
```



Problem 2:

2.1:

In [59]:

```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml

X = np.genfromtxt('X_train.txt', delimiter=',')
Y = np.genfromtxt('Y_train.txt', delimiter=',')
X,Y = ml.shuffleData(X,Y)
X = X[:,:41]
x1 = X[:,0]
print("First data point:")
print("Max:")
print(max(x1))
print("Min:")
print(min(x1))
print("Mean:")
print(np.mean(x1))
print("Variance:")
print(np.var(x1))
x2 = X[:,1]
print("Second data point:")
print("Max:")
```

```
print(max(x2))
print("Min:")
print(min(x2))
print("Mean:")
print(np.mean(x2))
print("Variance:")
print(np.var(x2))
x3 = X[:,2]
print("Three data point:")
print("Max:")
print(max(x3))
print("Min:")
print(min(x3))
print("Mean:")
print(np.mean(x3))
print("Variance:")
print(np.var(x3))
x4 = X[:,3]
print("Four data point:")
print("Max:")
print(max(x4))
print("Min:")
print(min(x4))
print("Mean:")
print(np.mean(x4))
print("Variance:")
print(np.var(x4))
x5 = X[:,4]
print("Five data point:")
print("Max:")
print(max(x5))
print("Min:")
print(min(x5))
print("Mean:")
print(np.mean(x5))
print("Variance:")
print(np.var(x5))
```

```
First data point:
Max:
110285.0
Min:
0.0
Mean:
1321.117413444699
Variance:
6747189.595085322
Second data point:
Max:
35.0
Min:
0.0
Mean:
6.5916745251246125
Variance:
34.70690630279573
Three data point:
Max:
51536.0
Min:
0.0
Mean:
1152.273237235619
Variance:
5376518.288798101
Four data point:
Max:
21768.0
Min:
0.0
Mean:
234.8262548834703
Variance:
260120.8305329767
Five data point:
Max:
27210.0
```

```
27210.0
Min:
0.0
Mean:
289.75871211100633
Variance:
406615.8651128233
```

2.2:

In [19]:

```
Xtr,Xva,Ytr,Yva = ml.splitData(X,Y,.50);
learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=50)
errVal = learner.err(Xva, Yva)
errTrain = learner.err(Xtr, Ytr)
print(errVal)
print(errTrain)
```

```
0.40932363244408515
0.0
```

2.3:

In [30]:

```
t, v = [],[]
for i in range(0,16):
    depth = i
    learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=i)
    a = learner.err(Xtr, Ytr)
    b = learner.err(Xva, Yva)
    t.append(a)
    v.append(b)
    print("Depth          Error on Training:        Error on Validation:")
    print(str(i) + "              " + str(a) + "      " + str(b))
```

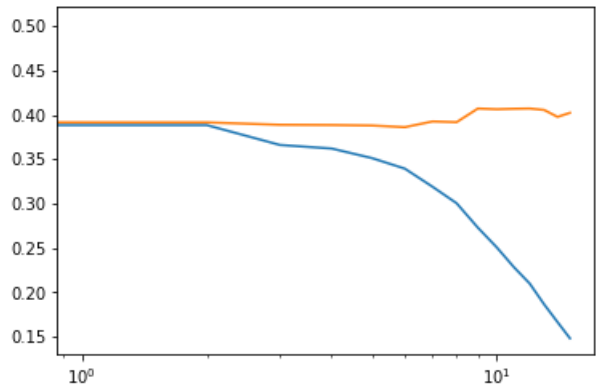
```
Depth          Error on Training:        Error on Validation:
0              0.48410560344827586      0.5049851791969819
Depth          Error on Training:        Error on Validation:
1              0.3884698275862069      0.39153866882241983
Depth          Error on Training:        Error on Validation:
2              0.3884698275862069      0.39153866882241983
Depth          Error on Training:        Error on Validation:
3              0.3661099137931034      0.38884397736459175
Depth          Error on Training:        Error on Validation:
4              0.3620689655172414      0.38857450821880896
Depth          Error on Training:        Error on Validation:
5              0.35129310344827586      0.3880355699272433
Depth          Error on Training:        Error on Validation:
6              0.3394396551724138      0.3861492859067637
Depth          Error on Training:        Error on Validation:
7              0.31896551724137934      0.3926165454055511
Depth          Error on Training:        Error on Validation:
8              0.3003771551724138      0.39180813796820263
Depth          Error on Training:        Error on Validation:
9              0.2728987068965517      0.40716787927782266
Depth          Error on Training:        Error on Validation:
10              0.2505387931034483      0.4063594718404743
Depth          Error on Training:        Error on Validation:
11              0.22817887931034483      0.40689841013203987
Depth          Error on Training:        Error on Validation:
12              0.20985991379310345      0.40716787927782266
Depth          Error on Training:        Error on Validation:
13              0.18642241379310345      0.4055510644031258
Depth          Error on Training:        Error on Validation:
14              0.16648706896551724      0.39773645917542444
Depth          Error on Training:        Error on Validation:
15              0.14816810344827586      0.40231743465373215
```

In [31]:

```
fig, ax = plt.subplots()
ax.semilogx(t)
ax.semilogx(v)
```

Out[31]:

```
[<matplotlib.lines.Line2D at 0x11cf813d0>]
```



Complexity correlates to depth. It becomes more complex as depth increases. Depth 6 seems to provide the best model because it has the lowest error.

2.4:

In [52]:

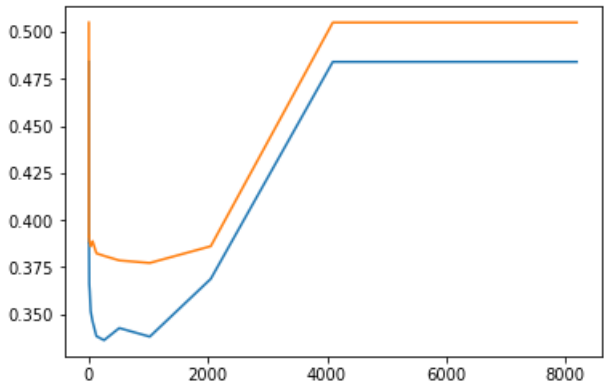
```
t, v = [],[]
for i in range(0,14):
    depth = i
    mp = 2**depth
    learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=i, minParent = mp)
    a = learner.err(Xtr, Ytr)
    b = learner.err(Xva, Yva)
    t.append(a)
    v.append(b)
    print("minParent   Depth          Error on Training:        Error on Validation:")
    print(str(mp)+"           "+str(i) + "              " + str(a) + "       " + str(b))
fig, ax = plt.subplots()
ax.plot([1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192],t)
ax.plot([1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192],v)
```

```
minParent   Depth          Error on Training:        Error on Validation:
1           0              0.48410560344827586      0.5049851791969819
minParent   Depth          Error on Training:        Error on Validation:
2           1              0.3884698275862069      0.39153866882241983
minParent   Depth          Error on Training:        Error on Validation:
4           2              0.3884698275862069      0.39153866882241983
minParent   Depth          Error on Training:        Error on Validation:
8           3              0.3661099137931034      0.38884397736459175
minParent   Depth          Error on Training:        Error on Validation:
16           4              0.3623383620689655      0.38830503907302616
minParent   Depth          Error on Training:        Error on Validation:
32           5              0.3515625      0.3861492859067637
minParent   Depth          Error on Training:        Error on Validation:
64           6              0.34617456896551724      0.38884397736459175
minParent   Depth          Error on Training:        Error on Validation:
128           7              0.33836206896551724      0.38237671786580435
minParent   Depth          Error on Training:        Error on Validation:
256           8              0.33620689655172414      0.3810293721368903
minParent   Depth          Error on Training:        Error on Validation:
512           9              0.3426724137931034      0.3786041498248451
minParent   Depth          Error on Training:        Error on Validation:
1024          10              0.3380926724137931      0.37725680409593104
minParent   Depth          Error on Training:        Error on Validation:
2048          11              0.3688038793103448      0.3861492859067637
minParent   Depth          Error on Training:        Error on Validation:
4096          12              0.48410560344827586      0.5049851791969819
minParent   Depth          Error on Training:        Error on Validation:
```

```
8192            13              0.48410560344827586      0.5049851791969819
```

Out[52]:

```
[<matplotlib.lines.Line2D at 0x11ac394d0>]
```



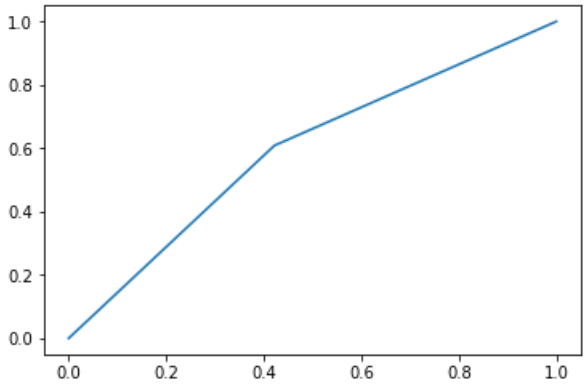
I think the complexity for min parent is hight when it reaches certain depths such as depth 10. It seems like depth 10 provides the best decision tree model.

2.6:

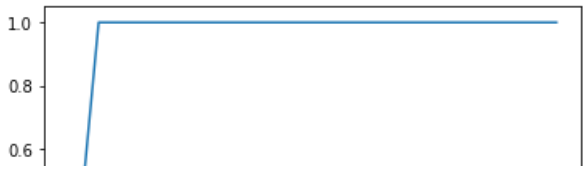
In [48]:

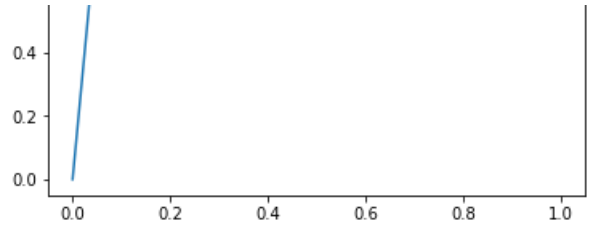
```
learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=50)
print("ROC Curve for Validation Points")
r = learner.roc(Xva, Yva)
plt.plot(r[0], r[1])
plt.show()
print("ROC Curve for Training Points")
r2 = learner.roc(Xtr, Ytr)
plt.plot(r2[0], r2[1])
plt.show()
auc = learner.auc(Xva, Yva)
auc1 = learner.auc(Xtr, Ytr)
print("AUC for Validation:")
print(auc)
print("AUC for Training:")
print(auc1)
```

```
ROC Curve for Validation Points
```




```
ROC Curve for Training Points
```



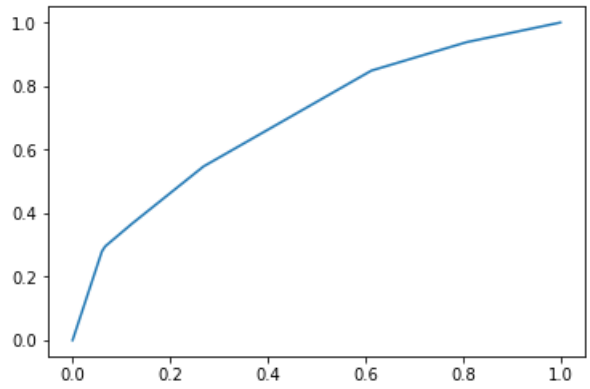



```
AUC for Validation:
0.5943487043570761
AUC for Training:
1.0
```

2.7:

In [57]:

```
minparent = 1024
maxdepth = 6
learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=maxdepth, minParent= minparent) # train a model
using training data X,Y
Xte = np.genfromtxt('X_test.txt', delimiter=',')
Yte = np.vstack((np.arange(Xte.shape[0]), learner.predictSoft(Xte)[:,1])).T # Output a file with tw
o columns, a row ID and a confidence in class 1:
np.savetxt('Y_submit.txt',Yte,'%d, %.2f',header='Id,Predicted',delimiter=',')
r = learner.roc(Xtr, Ytr)
plt.plot(r[0], r[1])
plt.show()
auc = learner.auc(Xtr, Ytr)
print(auc)
```




```
0.7015612036887706
```

My Kaggle username is timothyhakobian.

Problem 3:

In [63]:

```
e = []
for i in range(25):
    x,y  = ml.bootstrapData(Xtr,Ytr)
    e.append(ml.dtree.treeClassify(x, y, maxDepth = 15, minParent = 4, nFeatures = 60))
```

Statement of Collaboration: I did a google search on how to calculate entropy and came across this link: https://www.math.unipd.it/~aiolli/corsi/0708/IR/Lez12.pdf. I also looked a lot on piazza for help with problems which I thought I could use some clarification. I didn't use any other sources for help.