Name: Joyce Le
ID#: 82549113
UCI NetID: lejy

CS 178 Homework 4

Problem 1: Shattering and VC Dimension

1. This classifier can shatter (a) and (b), but not (c) or (d). This is because this classifier's decision boundary is a vertical line, and (c) and (d) do not pass a vertical line test if we label their points with alternating +1's and -1's.
2. This classifier can shatter (a), (b), and (c), but not (d). Instead of its decision boundary being a veritical line like in the previous problem, this classifier's decision boundary can be a line in any 2-D configuration. Therefore, it can also shatter (c) in addition to (a) and (b). It cannot shatter (d) because its VC-dimension is only 3, and (d) has four points.
3. This classifier can shatter (a), (b), and (c), but not (d). If we imagine a circle encompassing all the -1 points and excluding all the +1 points (or vice versa), then it is possible to shatter the data for (a), (b), and (c). However, for (d), if the points (2,2) and (8,6) are -1 and the other two points are +1, then we can not form a proper decision boundary because there would be no way for our circle to avoid encompassing the point (6,4).

Problem 2: Decision Trees for Spam Classification

1. $$\begin{aligned} p_y(1) &= 0.4 \\ 1 - p_y(1) &= 0.6 \\ H(y) &= -p \log_2 p - (1-p)\log_2(1-p) \\ &= -0.4 \log_2 0.4 - 0.6 \log_2 0.6 \\ &= 0.971 \end{aligned}$$

2. (see code below)

In [100]:

```
import numpy as np

dataset = np.array(
     [[0,0,1,1,0,-1],
      [1,1,0,1,0,-1],
      [0,1,1,1,1,-1],
      [1,1,1,1,0,-1],
      [0,1,0,0,0,-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],
      [1,1,1,1,1,-1]])

x = dataset[:,0:-1]
y = dataset[:,-1]

def H(feature):
    p = np.mean(feature > 0)
    if p == 0 or p == 1:
        return 0
    h = -p * np.log2(p) - (1-p) * np.log2(1-p)
    return h

def information_gain(feature, y):
    entropy = H(y)
    p = np.mean(feature)
    if p == 0 or p == 1:
        return 0
    # H(y | feature)
    ent = p * H(y[feature == 1]) + (1-p) * H(y[feature == 0])
    return entropy - ent

print("Information gain of\n")
print("x1:", information_gain(x[:,0], y))
print("x2:", information_gain(x[:,1], y))
print("x3:", information_gain(x[:,2], y))
print("x4:", information_gain(x[:,3], y))
print("x5:", information_gain(x[:,4], y))

print("\nYou should split on feature x2 for the root node because it has th
e highest information gain.")
```

```
Information gain of

x1: 0.04643934467101544
x2: 0.6099865470109874
x3: 0.0058021490143456145
x4: 0.09127744624168
x5: 0.0058021490143456145

You should split on feature x2 for the root node because it has the highest
information gain.
```

3.

In [22]:
```
# Split feature 2
print("Split on feature 2:")
print("Left:", dataset[x[:,1] == 0,:], "\n")
print("Right:", dataset[x[:,1] == 1,:])

print("\nRight side of data does not need to be further split, but left sid
e does.")
```

```
Split on feature 2:
Left: [[ 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]]

Right: [[ 1  1  0  1  0 -1]
 [ 0  1  1  1  1 -1]
 [ 1  1  1  1  0 -1]
 [ 0  1  0  0  0 -1]
 [ 1  1  1  1  1 -1]]

Right side of data does not need to be further split, but left side does.
```

In [33]:
```
print("Information gain of left side:")

left_dataset = dataset[x[:,1] == 0,:]
left = left_dataset[:,:-1]
right = left_dataset[:,-1]

print("x1:", information_gain(left[:,0], right))
print("x2:", information_gain(left[:,1], right))
print("x3:", information_gain(left[:,2], right))
print("x4:", information_gain(left[:,3], right))
print("x5:", information_gain(left[:,4], right))
```

```
Information gain of left side:
x1: 0.3219280948873623
x2: 0
x3: 0.07290559532005603
x4: 0.17095059445466865
x5: 0.07290559532005603
```

In [40]:
```
print("\nSplit on x1 next because it has the highest information gain:\n")

print("Left:", left_dataset[left[:,0] == 0,:], "\n")
print("Right:", left_dataset[left[:,0] == 1,:])

print("\nRight side does not need to be further split, but left side does."
)
print("X4 is the only feature that can be used to split left side.")
```

```
Split on x1 next because it has the highest information gain:

Left: [[ 0  0  1  1  0 -1]
 [ 0  0  1  0  0  1]]

Right: [[1 0 1 1 1 1]
 [1 0 0 0 0 1]
 [1 0 1 1 0 1]]

Right side does not need to be further split, but left side does.
X4 is the only feature that can be used to split left side.
```

In [95]:
```
print("Final decision tree:")

tree = '''
if (is long):
    discard
else:
    if (know author):
        read
    else:
        if (has 'grade'):
            discard
        else:
            read
'''

print(tree)
```

```
Final decision tree:

if (is long):
    discard
else:
    if (know author):
        read
    else:
        if (has 'grade'):
            discard
        else:
            read
```

Problem 3: Decision Trees on Kaggle

In [101]:
```
import mltools as ml

X = np.genfromtxt('all/X_train.txt', delimiter=None)
Y = np.genfromtxt('all/Y_train.txt', delimiter=None)
X,Y = ml.shuffleData(X,Y)

print("1.\n")
for i in range(14):
    print("Feature " + str(i + 1) + ":")
    print("min:", np.min(X[:,i]))
    print("max:", np.max(X[:,i]))
    print("mean:", np.mean(X[:,i]))
    print("variance:", np.var(X[:,i]))
    print()
```

```
1.

Feature 1:
min: 193.0
max: 253.0
mean: 241.58774300000002
variance: 83.95080688795099

Feature 2:
min: 152.5
max: 248.0
mean: 227.3859329
variance: 92.29796657769761

Feature 3:
min: 214.25
max: 252.38
mean: 241.56281370000002
variance: 35.300050500092304

Feature 4:
min: 152.5
max: 252.38
mean: 232.8259432
variance: 97.44001258437379

Feature 5:
min: 10.0
max: 31048.0
mean: 3081.98073
variance: 15614364.730518667

Feature 6:
min: 0.0
max: 13630.0
mean: 920.77918
variance: 3020579.876458528

Feature 7:
min: 0.0
max: 9238.0
mean: 137.15228
variance: 435636.57277080155

Feature 8:
min: 0.0
max: 125.17
mean: 3.2591865429
variance: 8.244616324135894

Feature 9:
min: 1.1031
max: 18.336
mean: 6.49802626
variance: 6.405266209771212

Feature 10:
```

```
min: 0.0
max: 13.23
mean: 2.09374870081
variance: 4.371492161137474

Feature 11:
min: 0.0
max: 21.89
mean: 4.220945725999999
variance: 4.040186886166696

Feature 12:
min: 0.0
max: 46.544
mean: 2.6914259745000004
variance: 2.1194924546399063

Feature 13:
min: 1.0221
max: 975.04
mean: 10.278704246999999
variance: 429.9989898718719

Feature 14:
min: -999.9
max: 782.5
mean: 5.705888000000001
variance: 3490.2878849314557
```

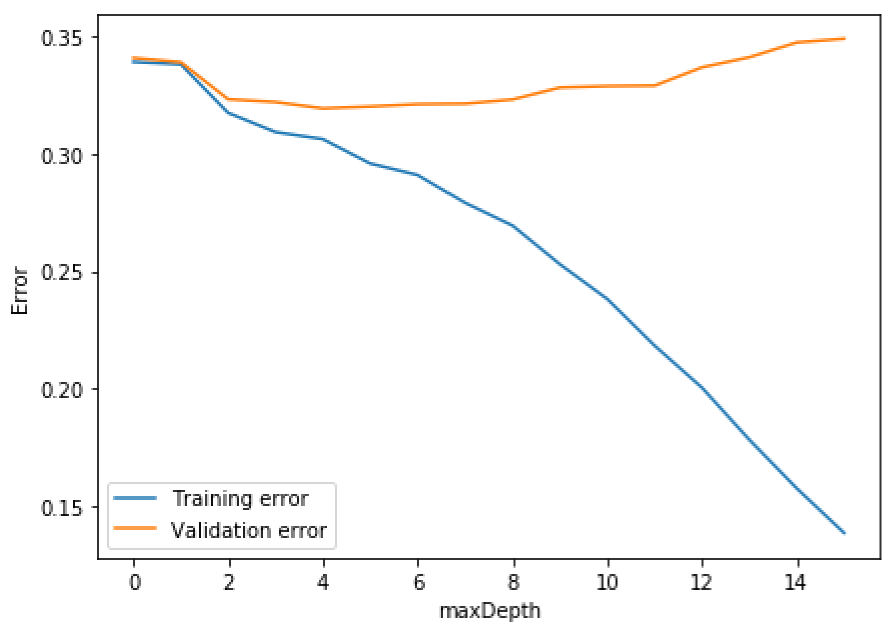
In [102]:

```
print("2.\n")
Xtr = X[:10000]
Ytr = Y[:10000]
Xva = X[10000:20000]
Yva = Y[10000:20000]

learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=50)
print("Training error:", learner.err(Xtr, Ytr))
print("Validation error:", learner.err(Xva, Yva))
```

```
2.

Training error: 0.0056
Validation error: 0.3819
```

In [103]:
```
import matplotlib.pyplot as plt

print("3.\n")

depth = np.arange(16)
training_error = []
validation_error = []

# get error rates
for x in depth:
    learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=x)
    training_error.append(learner.err(Xtr, Ytr))
    validation_error.append(learner.err(Xva, Yva))

# plot error rates vs. depth
f, ax = plt.subplots(1,1, figsize=(7,5))
ax.plot(depth, training_error, label = "Training error")
ax.plot(depth, validation_error, label = "Validation error")
ax.set_xlabel("maxDepth")
ax.set_ylabel("Error")
ax.legend()
plt.show()

print("Models with higher maxDepth have higher complexity.")
print("Best choice of maxDepth:", np.argmin(validation_error))
```

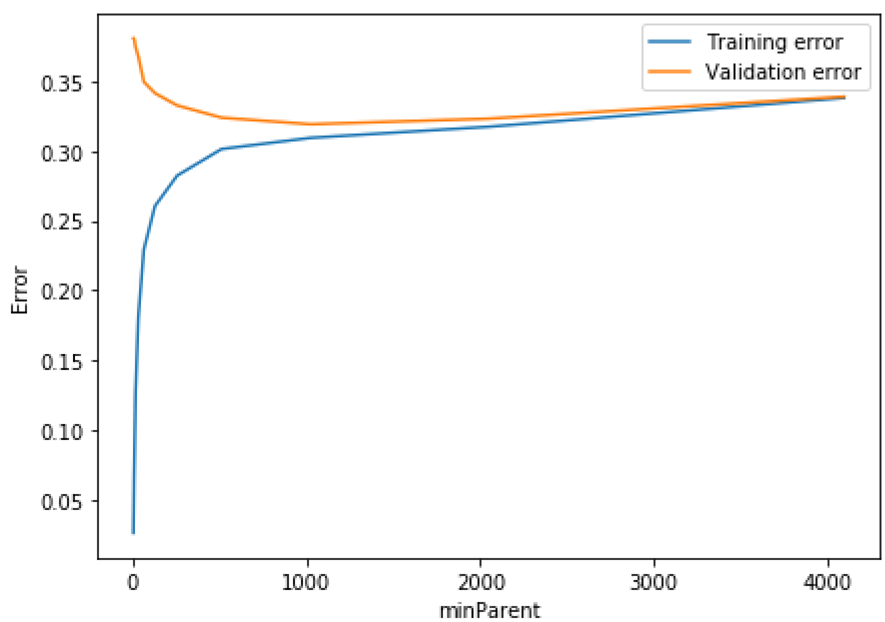
```
3.
```




```
Models with higher maxDepth have higher complexity.
Best choice of maxDepth: 4
```

In [104]:
```
print("4.\n")

minParent = 2 ** np.arange(2, 13)
training_error = []
validation_error = []

# get error rates
for x in minParent:
    learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth=50, minParent=x)
    training_error.append(learner.err(Xtr, Ytr))
    validation_error.append(learner.err(Xva, Yva))

# plot error rates vs. minParent
f, ax = plt.subplots(1,1, figsize=(7,5))
ax.plot(minParent, training_error, label = "Training error")
ax.plot(minParent, validation_error, label = "Validation error")
ax.set_xlabel("minParent")
ax.set_ylabel("Error")
ax.legend()
plt.show()

print("Models with higher minParent have lower complexity.")
print("Best choice of minParent:", minParent[np.argmin(validation_error)])
```

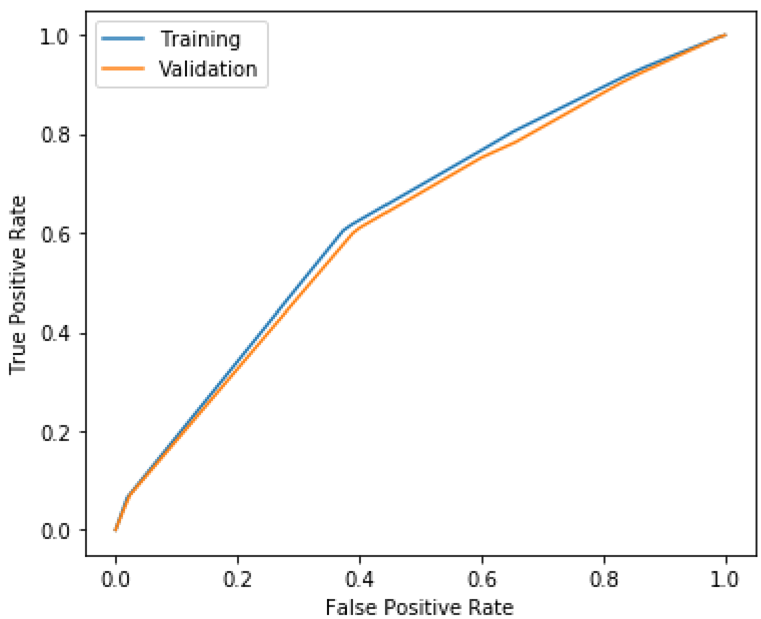
```
4.
```




```
Models with higher minParent have lower complexity.
Best choice of minParent: 1024
```

In [105]:
```
print("5.\n")

learner = ml.dtree.treeClassify(Xtr, Ytr, maxDepth = 4, minParent = 1024)

xroctr, yroctr, _ = learner.roc(Xtr, Ytr)
xrocva, yrocva, _ = learner.roc(Xva, Yva)

f, ax = plt.subplots(1, 1, figsize=(6,5))
ax.plot(xroctr, yroctr, label='Training')
ax.plot(xrocva, yrocva, label='Validation')
ax.set_xlabel("False Positive Rate")
ax.set_ylabel("True Positive Rate")
ax.legend()
plt.show()

print("Training AUC:", learner.auc(Xtr, Ytr))
print("Validation AUC:", learner.auc(Xva, Yva))
```

```
5.
```




```
Training AUC: 0.6518041846075654
Validation AUC: 0.6347389007802358
```

6.

In [107]:
```
learner = ml.dtree.treeClassify(X, Y, maxDepth=4, minParent=1024)
Xte = np.genfromtxt('all/X_test.txt', delimiter=None)
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,Prob1',comments='',deli
miter=',')
```

Kaggle Username: Joyce Le
Leaderboard AUC: 0.65058

Statement of Collaboration

I discussed the last part of Problem 2 with Ryan Sivoraphonh about how best to split the features of the dataset to create the decision tree. I also discussed with Ryan parts of Problem 3 about how to analyze the plots.