273ahw4

March 1, 2020

```
In [46]: import numpy as np
         import matplotlib.pyplot as plt
         import mltools as ml
         import math
```

0.0.1 Problem 1.1

```
In [47]: X = np.matrix([[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]])
```

```
In [48]: def calculate_entropy(value):
             p1 = np.mean(value>0)
             p2 = 1-p1
             if p1 ==0:
                 entropy = p2*np.log2(1/p2)
             elif p2==0 :
                 entropy = p1*np.log2(1/p1)
             else:
                 entropy = p1*np.log2(1/p1)+p2*np.log2(1/p2)
             return entropy
```

```
In [49]: Hy = calculate_entropy(X[:,-1])
         print('Entropy of class variable H(y) is %0.6f' % Hy)
```

```
Entropy of class variable H(y) is 0.970951
```

0.0.2 Problem 1.2

```
In [ ]:
```

```
In [50]: def calculate_info_gain(feature_index,X,entropy_class,feaure_lst):
             ture_pos = X[:,-1][X[:,feature_index-1]>0]
             entropy_pos = calculate_entropy(ture_pos)
             false_pos = X[:,-1][X[:,feature_index-1]<1]
             entropy_false = calculate_entropy(false_pos)
             p1 = np.mean(X[:,feature_index-1]>0)
```

```
            p2 = 1-p1
            info_gain = p1*(entropy_class-entropy_pos)+p2*(entropy_class-entropy_false)
            print('information gain for feature {} {} is {} \n'.format(feature_index,feaure_ls
            print('      entropy true  is: {} \n      entropy false is: {} \n'.format((entropy
            return info_gain
```

```
In [51]: feaure_lst = ['know author','is long','has research',
                       'has grade','has lottery']
         largest_info_index = 0
         H = 0
         for i in range(1,6):
             H_new = calculate_info_gain(i,X,Hy,feaure_lst);
             if H_new > H:
                 H = H_new
                 largest_info_index = i
         print('largest info index is {}'.format(largest_info_index))
```

```
information gain for feature 1 know author is 0.04643934467101556

      entropy true  is: 1.0
      entropy false is: 0.8112781244591328

information gain for feature 2 is long is 0.6099865470109875

      entropy true  is: 0.0
      entropy false is: 0.7219280948873623

information gain for feature 3 has research is 0.005802149014345906

      entropy true  is: 0.9852281360342514
      entropy false is: 0.9182958340544893

information gain for feature 4 has grade is 0.0912774462416802

      entropy true  is: 0.8631205685666309
      entropy false is: 0.9182958340544896

information gain for feature 5 has lottery is 0.0058021490143459024

      entropy true  is: 0.9182958340544893
      entropy false is: 0.9852281360342514

largest info index is 2
```

I should split feature 2 first, if email is long predict not read

0.0.3 Problem 1.3

```
In [52]: X = np.matrix([[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_new = np.delete(X,[1,2,3,4,9],0)
         X_new

Out[52]: matrix([[ 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]])

In [53]: def calculate_info_gain(feature_index,X,entropy_class,feaure_lst):
             ture_pos = X[:,-1][X[:,feature_index-1]>0]
             if ture_pos.size == 0:
                 entropy_pos = 0
             else:
                 entropy_pos = calculate_entropy(ture_pos)
             false_pos = X[:,-1][X[:,feature_index-1]<1]
             if false_pos.size ==0:
                 entropy_false = 0
             else:
                 entropy_false = calculate_entropy(false_pos)
             p1 = np.mean(X[:,feature_index-1]>0)
             p2 = 1-p1
             info_gain = p1*(entropy_class-entropy_pos)+p2*(entropy_class-entropy_false)
             print('information gain for new feature {} {} is {} \n'.format(feature_index,feau
             print('      entropy true  is: {} \n      entropy false is: {} \n'.format((entropy
             return info_gain

         feaure_lst = ['know author','has research',
                       'has grade','has lottery']
         largest_info_index = 0
         Hy = calculate_entropy(X[[0,5,3,7,8],-1])
         H = 0
         for i in range(1,5):
             H_new = calculate_info_gain(i,X_new,Hy,feaure_lst);
             if H_new > H:
                 H = H_new
                 largest_info_index = i
         print('largest info index is new/old feature {}'.format(largest_info_index))

information gain for new feature 1 know author is 0.5709505944546687

      entropy true  is: 0.0
      entropy false is: 1.0
```

```
information gain for new feature 2 has research is 0.24902249956730638

      entropy true  is: 0
      entropy false is: 0.7219280948873623

information gain for new feature 3 has grade is 0.3219280948873624

      entropy true  is: 0.8112781244591328
      entropy false is: 0.0

information gain for new feature 4 has lottery is 0.41997309402197497

      entropy true  is: 0.9182958340544896
      entropy false is: 0.0

largest info index is new/old feature 1
```

0.1 Decision Tree

```
In [55]: X_new_new = np.delete(X_new,[1,3,4],0)
         X_new_new = np.delete(X_new_new,[0,1],1)
         X_new_new;
```

0.1.1 Complete Desicion Tree (verbal version)

if X2(is long?) is True:

```
predict Not Read (since entropy is zero)
```

elif X2 is False:

```
if X1(know author?) is True:

    predict Read

elif X1 is False:
    # (note that we only have two raws now as X_new_new shows)
    if X4 (has grade) is True:

        predict not read

    elif X4  (has grade) is False:

        predict read
```

0.1.2 Complete Desicion Tree (number version)

if X2 == 1:

```
Y = 0
```

else:

```
if X1 == 1:

    Y = 1

else:

    if X4 == 1:

        Y = 0

    else:

        Y = 1
```

0.1.3 Problem 2.1

```
In [56]: import mltools as ml

In [57]: X = np.genfromtxt('data/X_train.txt',delimiter=None)
         Y = np.genfromtxt('data/Y_train.txt',delimiter=None)
         # X,Y = ml.shuffleData(X,Y)

In [62]: print('already shuffled')
         Y[0:100]

already shuffled


Out[62]: array([1., 0., 0., 0., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 0., 0.,
                0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1.,
                0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 1., 0., 1., 0., 0., 0.,
                0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0., 1., 1., 0., 0., 1., 0.,
                0., 1., 1., 1., 0., 0., 1., 1., 0., 1., 1., 0., 0., 0., 0.])

In [63]: for i in range(X.shape[1]):
             print("feature {} minimum  is {}".format(i+1,np.min(X[:,i])))
             print("feature {} maximum  is {}".format(i+1,np.max(X[:,i])))
             print("feature {} mean     is {}".format(i+1,np.mean(X[:,i])))
             print("feature {} variance is {}\n".format(i+1,np.var(X[:,i])))
```

```
feature 1 minimum  is 197.0
feature 1 maximum  is 253.0
feature 1 mean     is 241.89897349999998
feature 1 variance is 81.19881598129776

feature 2 minimum  is 190.0
feature 2 maximum  is 248.0
feature 2 mean     is 228.38130700000002
feature 2 variance is 89.150265341751

feature 3 minimum  is 214.97
feature 3 maximum  is 252.02
feature 3 mean     is 241.90593450000003
feature 3 variance is 34.55774434670975

feature 4 minimum  is 205.42
feature 4 maximum  is 252.02
feature 4 mean     is 233.8253765
feature 4 variance is 94.50721140824776

feature 5 minimum  is 10.0
feature 5 maximum  is 17130.0
feature 5 mean     is 2849.0465
feature 5 variance is 10505588.30063775

feature 6 minimum  is 0.0
feature 6 maximum  is 12338.0
feature 6 mean     is 862.8611
feature 6 variance is 3090415.2075067903

feature 7 minimum  is 0.0
feature 7 maximum  is 9238.0
feature 7 mean     is 163.65265
feature 7 variance is 698073.3556979776

feature 8 minimum  is 0.0
feature 8 maximum  is 27.419
feature 8 mean     is 3.0557549345
feature 8 variance is 7.276890946708305

feature 9 minimum  is 1.2189
feature 9 maximum  is 18.107
feature 9 mean     is 6.311441945
feature 9 variance is 6.183003202965116

feature 10 minimum  is 0.0
feature 10 maximum  is 11.368
feature 10 mean     is 1.8939148043499998
```

```
feature 10 variance is 4.150931810214395

feature 11 minimum  is 0.0
feature 11 maximum  is 18.771
feature 11 mean     is 4.289551351
feature 11 variance is 3.944615292538295

feature 12 minimum  is 0.0
feature 12 maximum  is 14.745
feature 12 mean     is 2.7977508345000004
feature 12 variance is 1.9323439727669185

feature 13 minimum  is 1.0271
feature 13 maximum  is 278.71
feature 13 mean     is 10.452536635
feature 13 variance is 170.00184292005338

feature 14 minimum  is -999.9
feature 14 maximum  is 769.2
feature 14 mean     is 7.65813
feature 14 variance is 1528.9473589030997
```

```
In [64]: import pandas as pd
         import os
         new_dataframe = pd.DataFrame(
             {
                 "feature index" :range(1,15),
                 "minimum":np.min(X,0),
                 "maximum":np.max(X,0),
                 "mean":np.mean(X,0),
                 "var": np.var(X,0)
             },
             index = ['']*len(X[0])
         )
         new_dataframe
```

Out[64]:

feature index	minimum	maximum	mean	var
1	197.0000	253.000	241.898974	8.119882e+01
2	190.0000	248.000	228.381307	8.915027e+01
3	214.9700	252.020	241.905935	3.455774e+01
4	205.4200	252.020	233.825377	9.450721e+01
5	10.0000	17130.000	2849.046500	1.050559e+07
6	0.0000	12338.000	862.861100	3.090415e+06
7	0.0000	9238.000	163.652650	6.980734e+05
8	0.0000	27.419	3.055755	7.276891e+00
9	1.2189	18.107	6.311442	6.183003e+00

```
10    0.0000     11.368     1.893915  4.150932e+00
11    0.0000     18.771     4.289551  3.944615e+00
12    0.0000     14.745     2.797751  1.932344e+00
13    1.0271    278.710    10.452537  1.700018e+02
14 -999.9000    769.200     7.658130  1.528947e+03
```

0.1.4 Problem 2.2

```
In [69]: Xtr = X[:10000] # shuffled
         Ytr = Y[:10000]
         Xva = X[10000:20000]
         Yva = Y[10000:20000]
         learner = ml.dtree.treeClassify(Xtr,Ytr,maxDepth=50)

In [70]: print('Depth 50 Training   error is:  {}'.format(learner.err(Xtr,Ytr)))
         print('Depth 50 Validation error is:  {}'.format(learner.err(Xva,Yva)))

Depth 50 Training   error is:  0.0098
Depth 50 Validation error is:  0.3801
```

0.1.5 Problem 2.3

```
In [71]: n = 16
         Train_err_lst = np.zeros(len(range(0,n)))
         Vladi_err_lst = np.zeros(len(range(0,n)))

In [72]: for i in range(len(range(0,n))):
             learner = ml.dtree.treeClassify(Xtr,Ytr,maxDepth=i)
             Train_err_lst[i]=learner.err(Xtr,Ytr)
             Vladi_err_lst[i]=learner.err(Xva,Yva)

In [73]: fig,ax = plt.subplots(1,1,figsize=(9,8))
         ax.plot(range(0,n),Train_err_lst,'ro-')
         ax.plot(range(0,n),Vladi_err_lst,'bo-')
         ax.legend(["Traning Error Rate","Validation Error Rate"])
         ax.set_title("Error Rates versus maxDepth",fontsize=15)

Out[73]: Text(0.5,1,'Error Rates versus maxDepth')
```


```
In [79]: new_dataframe = pd.DataFrame(
             {
                 "maxDepth": range(0,16),
                 "Training Error" :Train_err_lst,
                 "Validation Error":Vladi_err_lst
             },
             index = ['']*len(range(0,16))
         )
         new_dataframe
```

Out[79]:

	maxDepth	Training Error	Validation Error
	0	0.3645	0.3696
	1	0.3358	0.3429
	2	0.3247	0.3333
	3	0.3232	0.3326
	4	0.3114	0.3197
	5	0.3035	0.3226

6	0.2918	0.3330
7	0.2809	0.3232
8	0.2726	0.3237
9	0.2545	0.3332
10	0.2391	0.3403
11	0.2234	0.3395
12	0.2039	0.3446
13	0.1802	0.3466
14	0.1637	0.3501
15	0.1466	0.3543

Model with higer maxDepth has higher complexity.

Based on the Validation error MaxDepth = 4 gives the best decision tree model for value range 0,1,2, ..., 15

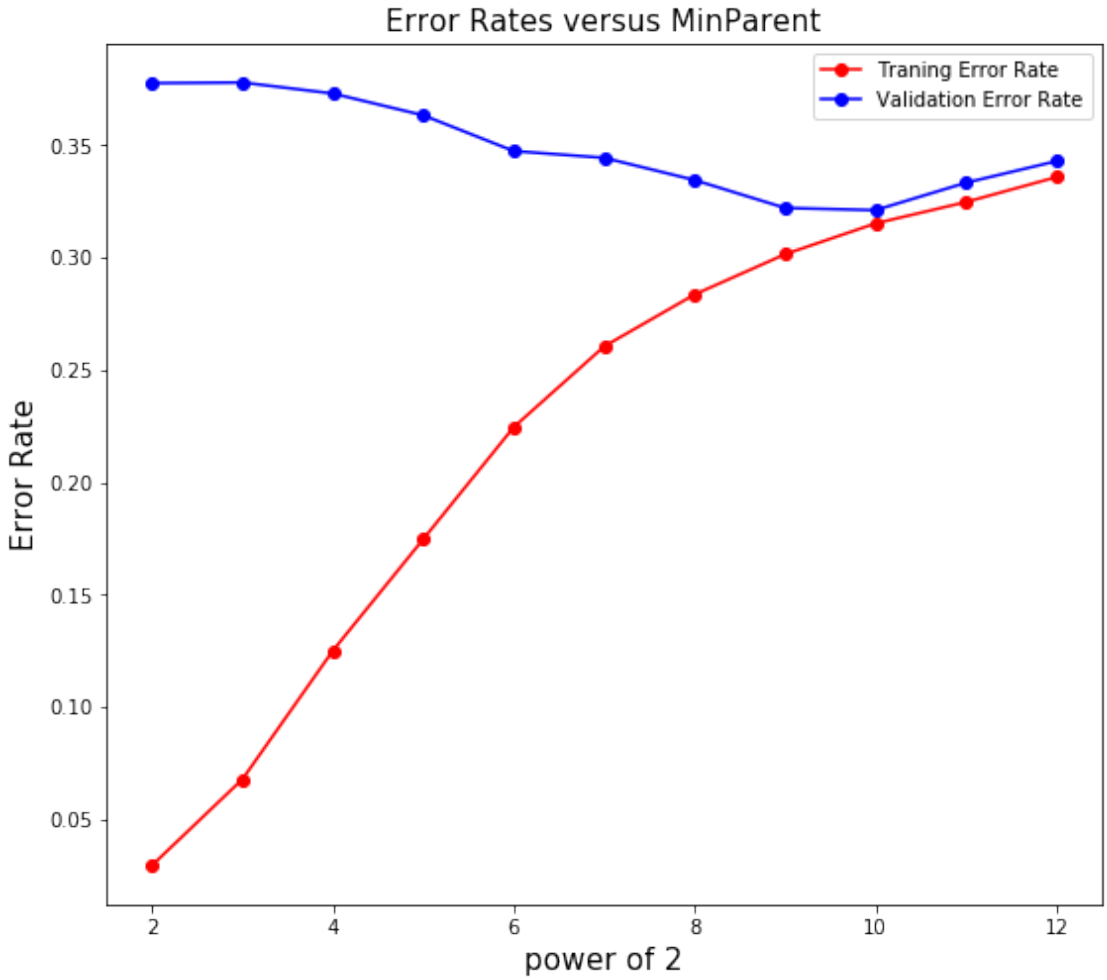
0.1.6 Problem 2.4

```
In [19]: n = 11
         Train_err_lst_pa = np.zeros(len(range(0,n)))
         Vladi_err_lst_pa = np.zeros(len(range(0,n)))

         for i in range(len(range(0,n))):
             learner = ml.dtree.treeClassify(Xtr,Ytr,maxDepth=50,minParent = 2**(i+2))
             Train_err_lst_pa[i]=learner.err(Xtr,Ytr)
             Vladi_err_lst_pa[i]=learner.err(Xva,Yva)

In [20]: fig,ax = plt.subplots(1,1,figsize=(9,8))
         ax.plot(range(2,n+2),Train_err_lst_pa,'ro-')
         ax.plot(range(2,n+2),Vladi_err_lst_pa,'bo-')
         ax.legend(["Traning Error Rate","Validation Error Rate"])
         ax.set_title("Error Rates versus MinParent",fontsize=15)
         plt.xlabel("power of 2",fontsize=15)
         plt.ylabel("Error Rate",fontsize=15)

Out[20]: Text(0,0.5,'Error Rate')
```




```
In [81]: new_dataframe = pd.DataFrame(
             {
                 "MiniParent Order of 2": range(2,13),
                 "Training Error" :Train_err_lst_pa,
                 "Validation Error":Vladi_err_lst_pa
             },
             index = ['']*len(range(2,13))
         )
         new_dataframe
```

Out[81]:

MiniParent Order of 2	Training Error	Validation Error
2	0.0297	0.3776
3	0.0680	0.3779
4	0.1252	0.3730
5	0.1748	0.3633
6	0.2246	0.3474
7	0.2605	0.3444

8	0.2836	0.3345
9	0.3015	0.3221
10	0.3152	0.3211
11	0.3247	0.3333
12	0.3358	0.3429

Models with higher minParent has lower complexity.
I will choose minParent = 2^10 for the least validation error as the best decision tree model.

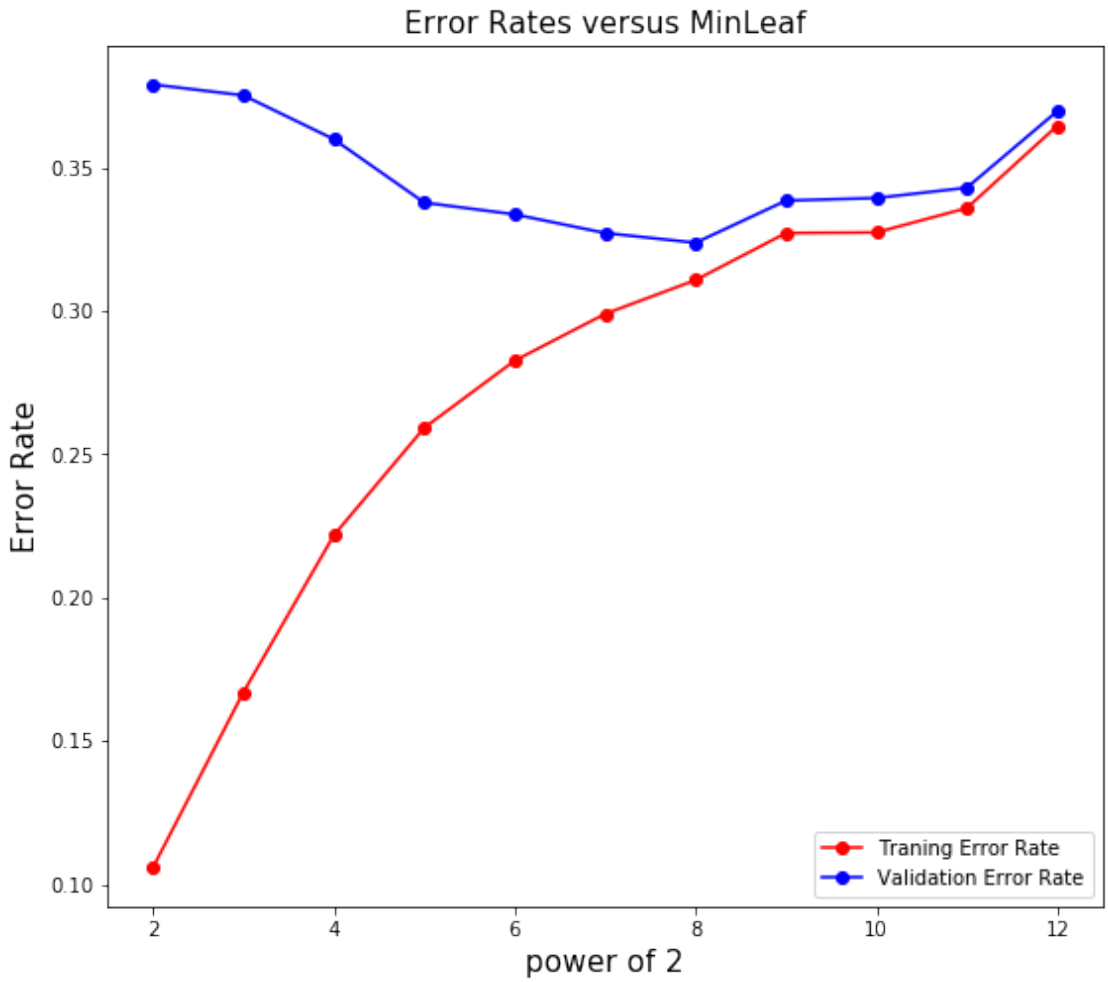
0.1.7 Problem 2.5

In [82]:
```
n = 11
Train_err_lst_le = np.zeros(len(range(0,n)))
Vladi_err_lst_le = np.zeros(len(range(0,n)))

for i in range(len(range(0,n))):
    learner = ml.dtree.treeClassify(Xtr,Ytr,maxDepth=50,minLeaf = 2**(i+2))
    Train_err_lst_le[i]=learner.err(Xtr,Ytr)
    Vladi_err_lst_le[i]=learner.err(Xva,Yva)

fig,ax = plt.subplots(1,1,figsize=(9,8))
ax.plot(range(2,n+2),Train_err_lst_le,'ro-')
ax.plot(range(2,n+2),Vladi_err_lst_le,'bo-')
ax.legend(["Traning Error Rate","Validation Error Rate"])
ax.set_title("Error Rates versus MinLeaf",fontsize=15)
plt.xlabel("power of 2",fontsize=15)
plt.ylabel("Error Rate",fontsize=15)
```

Out[82]:
```
Text(0,0.5,'Error Rate')
```



0.1.8 Problem 2.6

```
In [84]: print("Minumum for Depth     control is :{}".format(np.min(Vladi_err_lst)))
         print("Minumum for MinParent control is :{}".format(np.min(Vladi_err_lst_pa)))
         print("Minumum for MinLeaf   control is :{}".format(np.min(Vladi_err_lst_le)))
```

```
Minumum for Depth     control is :0.3197
Minumum for MinParent control is :0.3211
Minumum for MinLeaf   control is :0.3237
```

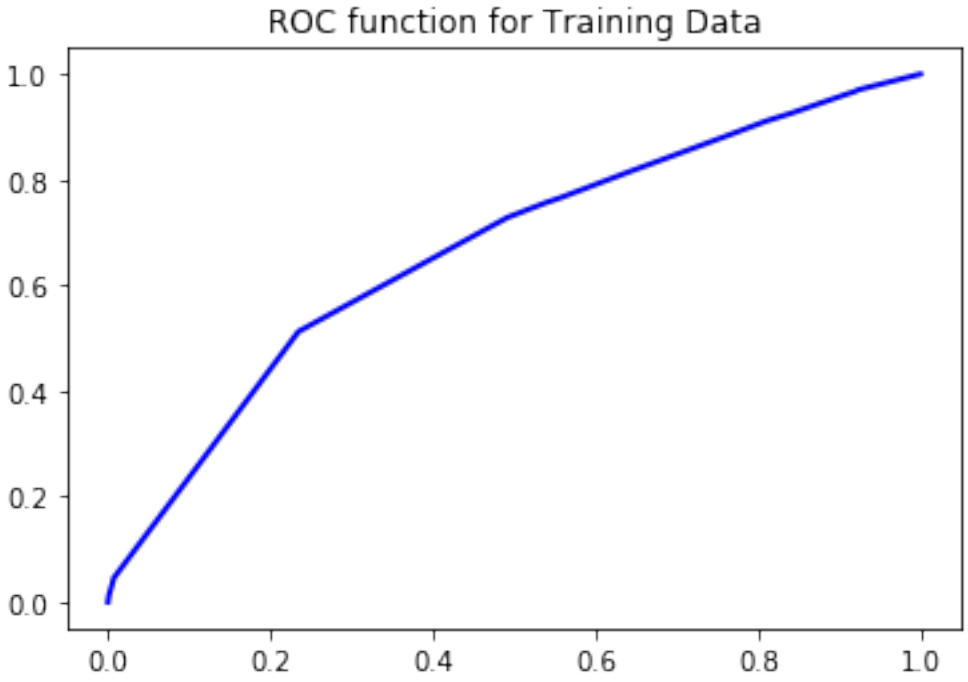
Best Train Model is from Min Depth = 4

```
In [ ]:
```

```
In [87]: learner = ml.dtree.treeClassify(Xtr,Ytr,maxDepth=4)
         fpr, tpr, tnr = learner.roc(Xtr, Ytr)
```

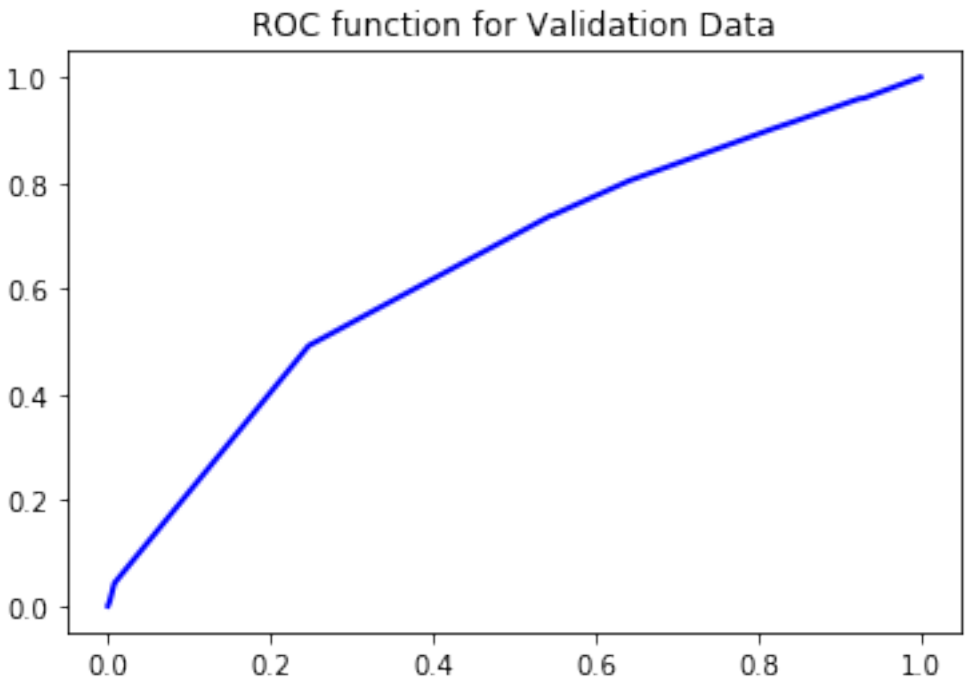
```
In [88]: plt.plot(fpr, tpr, 'b-', linewidth = 2)
         plt.title("ROC function for Training Data")

Out[88]: Text(0.5,1,'ROC function for Training Data')
```




```
In [91]: fpr, tpr, tnr = learner.roc(Xva, Yva)
         plt.plot(fpr, tpr, 'b-', linewidth = 2)
         plt.title("ROC function for Validation Data")

Out[91]: Text(0.5,1,'ROC function for Validation Data')
```




```
In [92]: print("AUC score for training   data is {}".format(learner.auc(Xtr, Ytr)))
         print("AUC score for Validation data is {}".format(learner.auc(Xva, Yva)))
```

```
AUC score for training   data is 0.6806329008730151
AUC score for Validation data is 0.6621804706899488
```

0.1.9 Problem 2.7

```
In [121]: learner = ml.dtree.treeClassify(Xtr,Ytr,maxDepth=4,minParent =2**8 )
          learner.err(Xtr,Ytr)
          learner.err(Xva,Yva)
```

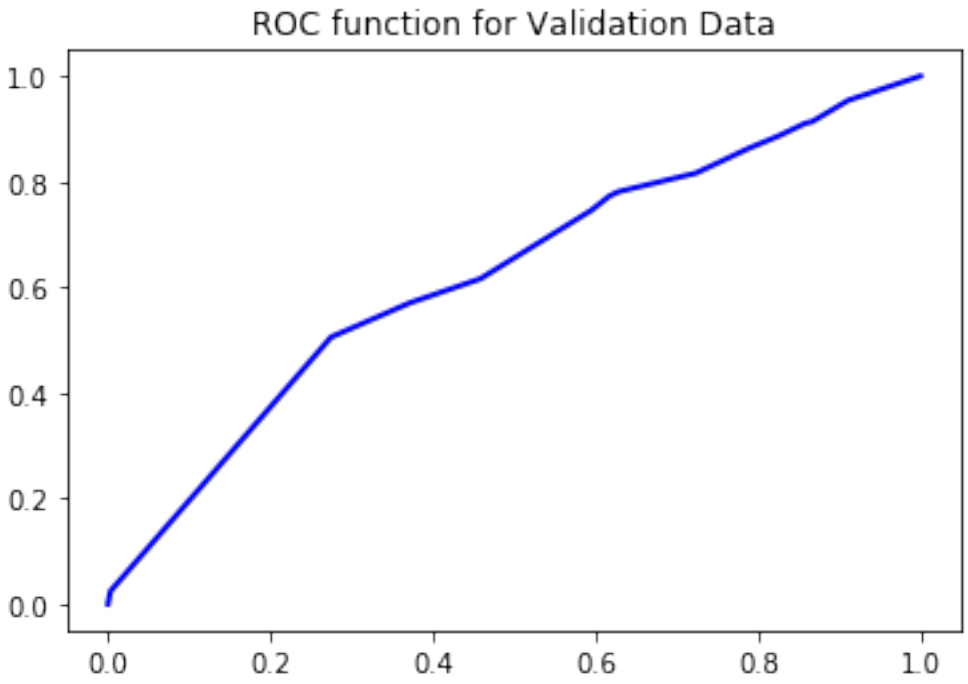
```
Out[121]: 0.3197
```

```
In [122]: X = np.genfromtxt('data/X_train.txt',delimiter=None)
          Y = np.genfromtxt('data/Y_train.txt',delimiter=None)
          # X,Y = ml.shuffleData(X,Y)
          Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.95)
```

```
In [123]: learner = ml.dtree.treeClassify(Xtr,Ytr,maxDepth=4,minParent =2**8 )
```

```
In [124]: fpr, tpr, tnr = learner.roc(Xte, Yte)
          plt.plot(fpr, tpr, 'b-', linewidth = 2)
          plt.title("ROC function for Validation Data")
```

```
Out[124]: Text(0.5,1,'ROC function for Validation Data')
```




```
In [126]: print('Accuracy rate is {}'.format(learner.err(Xte,Yte)))
          print("AUC score for Test data is {}".format(learner.auc(Xte, Yte)))

Accuracy rate is 0.332
AUC score for Test data is 0.639002574002574
```

0.2 Random Forests

0.2.1 Problem 3.1.1

```
In [137]: X = np.genfromtxt('data/X_train.txt',delimiter=None)
          Y = np.genfromtxt('data/Y_train.txt',delimiter=None)
          # X,Y = ml.shuffleData(X,Y)
          Xt ,Xte,Yt ,Yte = ml.splitData(X, Y, 0.9)
          Xtr,Xva,Ytr,Yva = ml.splitData(Xt,Yt,0.75)
          m,n = Xtr.shape

In [138]: nBag = 150
          classifiers = [None]*nBag
          for i in range(nBag):
              # ind = np.floor(m*np.random.rand(nUse)).astype(int)
              Xi,Yi = ml.bootstrapData(Xtr,Ytr,n_boot=10000)
              classifiers[i] = ml.dtree.treeClassify(Xi,Yi,maxDepth=16,minLeaf=4,nFeatures=8)
```

```
In [139]: mva = Xva.shape[0]
          mtr = Xtr.shape[0]
          predict_va = np.zeros((mva,nBag))
          predict_tr = np.zeros((mtr,nBag))
          for i in range(nBag):
              predict_va[:,i]=classifiers[i].predict(Xva)
              predict_tr[:,i]=classifiers[i].predict(Xtr)

In [140]: ensemble_num = range(0,nBag+1)
          err_va = [None]*len(ensemble_num)
          err_tr = [None]*len(ensemble_num)
          for i,num in enumerate(ensemble_num):
              predicti  = np.mean(predict_va[:,0:num],axis=1)>0.5
              err_va[i] = np.mean(predicti.reshape(Yva.shape)!=Yva)
              predicti  = np.mean(predict_tr[:,0:num],axis=1)>0.5
              err_tr[i] = np.mean(predicti.reshape(Ytr.shape)!=Ytr)
```

```
C:\Users\Yushang\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2920: RuntimeWarning: Me
  out=out, **kwargs)
C:\Users\Yushang\anaconda3\lib\site-packages\numpy\core\_methods.py:78: RuntimeWarning: invalid
  ret, rcount, out=ret, casting='unsafe', subok=False)
C:\Users\Yushang\anaconda3\lib\site-packages\ipykernel_launcher.py:5: RuntimeWarning: invalid v
  """
C:\Users\Yushang\anaconda3\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: invalid v
  import sys
```

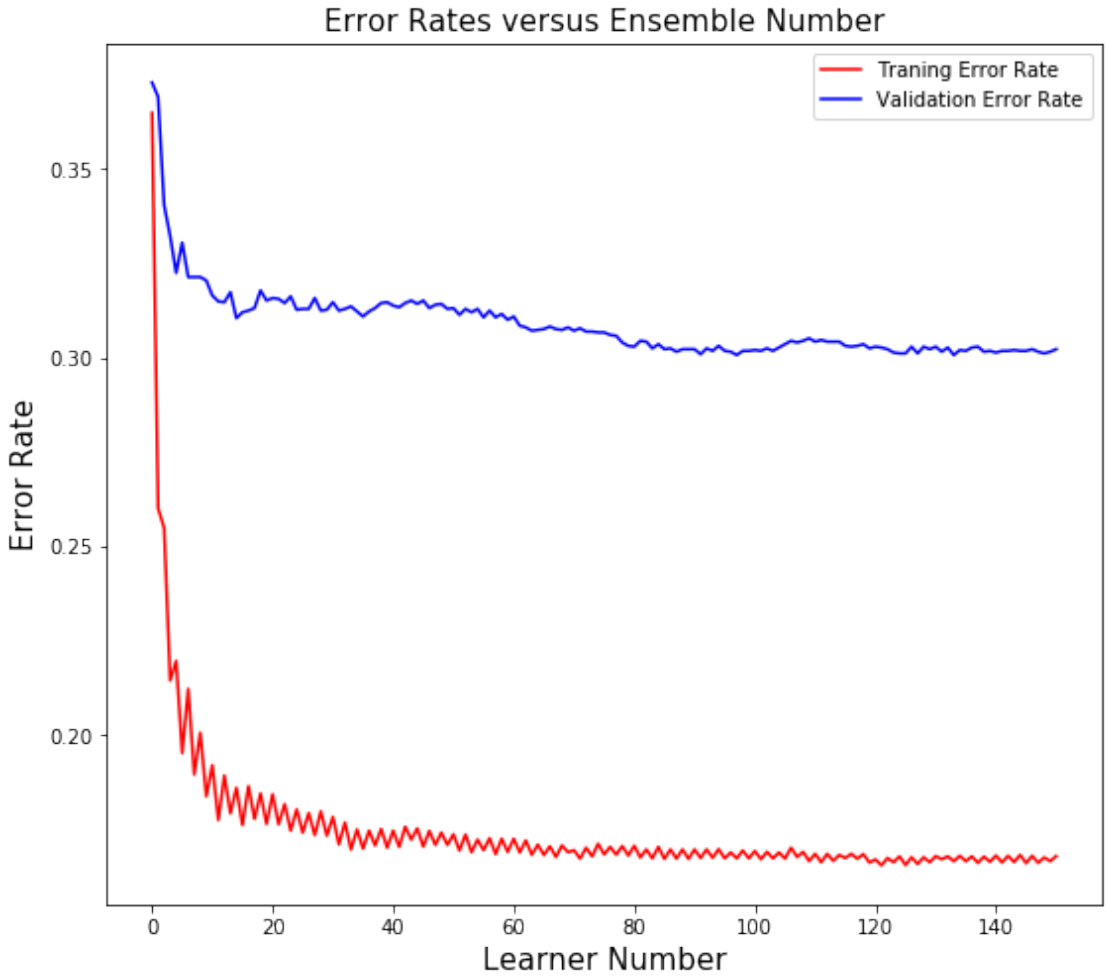
```
In [141]: np.argwhere(err_va==np.min(err_va)).flatten()

Out[141]: array([ 97, 133], dtype=int64)
```

0.2.2 Problem 3.2

```
In [143]: fig,ax = plt.subplots(1,1,figsize=(9,8))
          ax.plot(ensemble_num,err_tr,'r-')
          ax.plot(ensemble_num,err_va,'b-')
          ax.legend(["Traning Error Rate","Validation Error Rate"])
          ax.set_title("Error Rates versus Ensemble Number",fontsize=15)
          plt.xlabel("Learner Number",fontsize=15)
          plt.ylabel("Error Rate",fontsize=15)

Out[143]: Text(0,0.5,'Error Rate')
```




```
In [147]: new_dataframe = pd.DataFrame(
              {
                  "Learner Number": range(0,nBag+1),
                  "Training Error" :err_tr,
                  "Validation Error":err_va
              },
              index = ['']*len(range(0,nBag+1))
          )
          new_dataframe
```

Out[147]:

	Learner Number	Training Error	Validation Error
	0	0.364815	0.372889
	1	0.260074	0.369111
	2	0.254889	0.340444
	3	0.214444	0.332222
	4	0.219630	0.322444
	5	0.195259	0.330444

	6	0.212222	0.321333
	7	0.189556	0.321333
	8	0.200593	0.321333
	9	0.183778	0.320444
	10	0.192000	0.316444
	11	0.177481	0.314889
	12	0.189259	0.314667
	13	0.179259	0.317333
	14	0.186000	0.310444
	15	0.176148	0.312000
	16	0.186444	0.312444
	17	0.177778	0.313111
	18	0.184519	0.317778
	19	0.176444	0.315111
	20	0.184296	0.315778
	21	0.176370	0.315556
	22	0.181704	0.314444
	23	0.174667	0.316222
	24	0.180296	0.312667
	25	0.174148	0.312889
	26	0.179333	0.312889
	27	0.173481	0.315778
	28	0.179778	0.312444
	29	0.173333	0.312667
..	...	...	...
	121	0.165481	0.302667
	122	0.167407	0.302222
	123	0.166296	0.301333
	124	0.167852	0.301111
	125	0.165556	0.301111
	126	0.167630	0.302889
	127	0.165852	0.301111
	128	0.167630	0.302889
	129	0.166296	0.302222
	130	0.167926	0.302889
	131	0.167111	0.301556
	132	0.167852	0.302667
	133	0.166593	0.300667
	134	0.168000	0.302000
	135	0.166593	0.301778
	136	0.167852	0.302667
	137	0.166148	0.302889
	138	0.167778	0.301556
	139	0.166444	0.301778
	140	0.168074	0.301333
	141	0.166222	0.301778
	142	0.168000	0.301778
	143	0.166370	0.302000

```
     144        0.168222          0.301778
     145        0.166074          0.301778
     146        0.168000          0.302222
     147        0.166148          0.301556
     148        0.167556          0.301111
     149        0.166593          0.301556
     150        0.167926          0.302222

[151 rows x 3 columns]
```

0.2.3 Problem 3.3

```
In [148]: min_index = ensemble_num[np.argwhere(err_va==np.min(err_va)).flatten()[0]]
          min_index
```

```
Out[148]: 97
```

```
In [149]: Xi = Xtr[0:13500]
          Yi = Ytr[0:13500]
          nBag = min_index
          classifiers = [None]*nBag
          for i in range(nBag):
              # ind = np.floor(m*np.random.rand(nUse)).astype(int)
              Xi,Yi = ml.bootstrapData(X,Y,n_boot=4000)
              classifiers[i] = ml.dtree.treeClassify(Xi,Yi,maxDepth=16,minLeaf=4,nFeatures=8)
          mte = Xte.shape[0]
          predict_te = np.zeros((mte,nBag))
          for i in range(nBag):
              predict_te[:,i]=classifiers[i].predict(Xte)
          predict_te  = np.mean(predict_te,axis=1)>0.5
          err_te = np.mean(predict_te.reshape(Yte.shape)!=Yte)
```

```
In [151]: print(" Using 10k training data with ensmble size {}\n test error is: {} ".format(mi
          print('this error is much less than 0.3192 in problem 2')
```

```
 Using 10k training data with ensmble size 97
 test error is: 0.2465
this error is much less than 0.3192 in problem 2
```

This error rate is much smaller than that we got in Problem 2

0.3 Option 1 end

```
In [42]: #X = np.genfromtxt('data/X_train.txt',delimiter=None)
         #Y = np.genfromtxt('data/Y_train.txt',delimiter=None)
         #X,Y = ml.shuffleData(X,Y)
         #Xt ,Xte,Yt ,Yte = ml.splitData(X, Y, 0.9)
         #Xtr,Xva,Ytr,Yva = ml.splitData(Xt,Yt,0.75)
         #m,n = Xtr.shape
```

Modify TreeBase
def train(self, X, Y, minParent=2, maxDepth=np.inf, minLeaf=1, nFeatures=None):
->
def train(self, X, Y, minParent=2, maxDepth=3, minLeaf=1, nFeatures=None):

In [43]:
```
#from numpy import asarray as arr

#def delt(Xc):
#    return arr(1/(1+np.exp(-Xc)))
#def regressor(learner,Xtr,Ytr):
#    Xc = learner.predict(Xtr)
#    dY = np.zeros(Ytr.shape)
#   dY[Ytr==1] =1-delt(Xc[Ytr==1])
#    dY[Ytr!=1] = -delt(Xc[Ytr!=1])
#    return dY
```

In [44]:
```
#nBoost = 25
#learner = [None]*nBoost
#alpha = [1.0] *nBoost

#dY = Ytr-Ytr
#dY = 1/(1+np.exp(dY))
#
#for k in range(nBoost):
#    learner[k]=ml.dtree.treeRegress(Xtr,dY)
#    alpha[k] =1.0
 #   dY = dY -alpha[k]*learner[k].predict(Xtr)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

0.3.1 Problem 4

I did this hw independently

In [ ]: