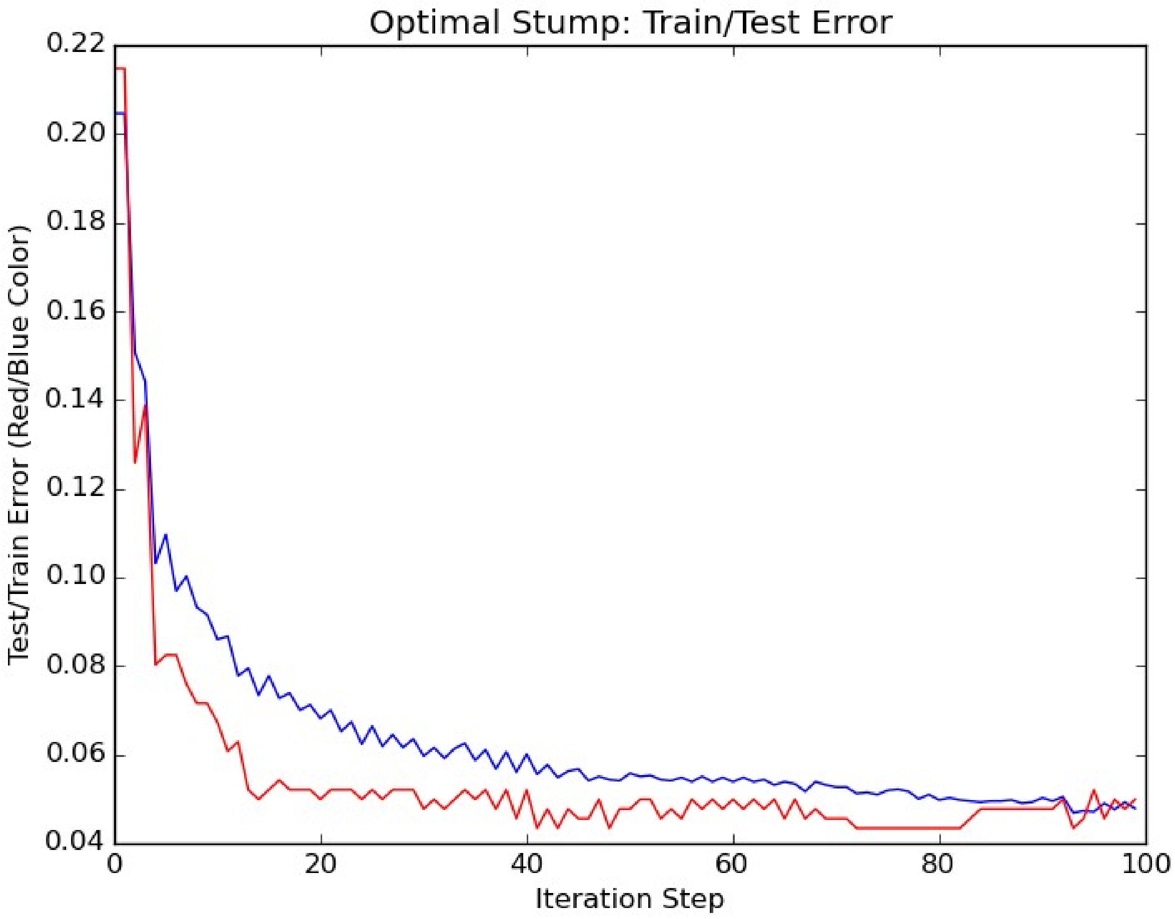
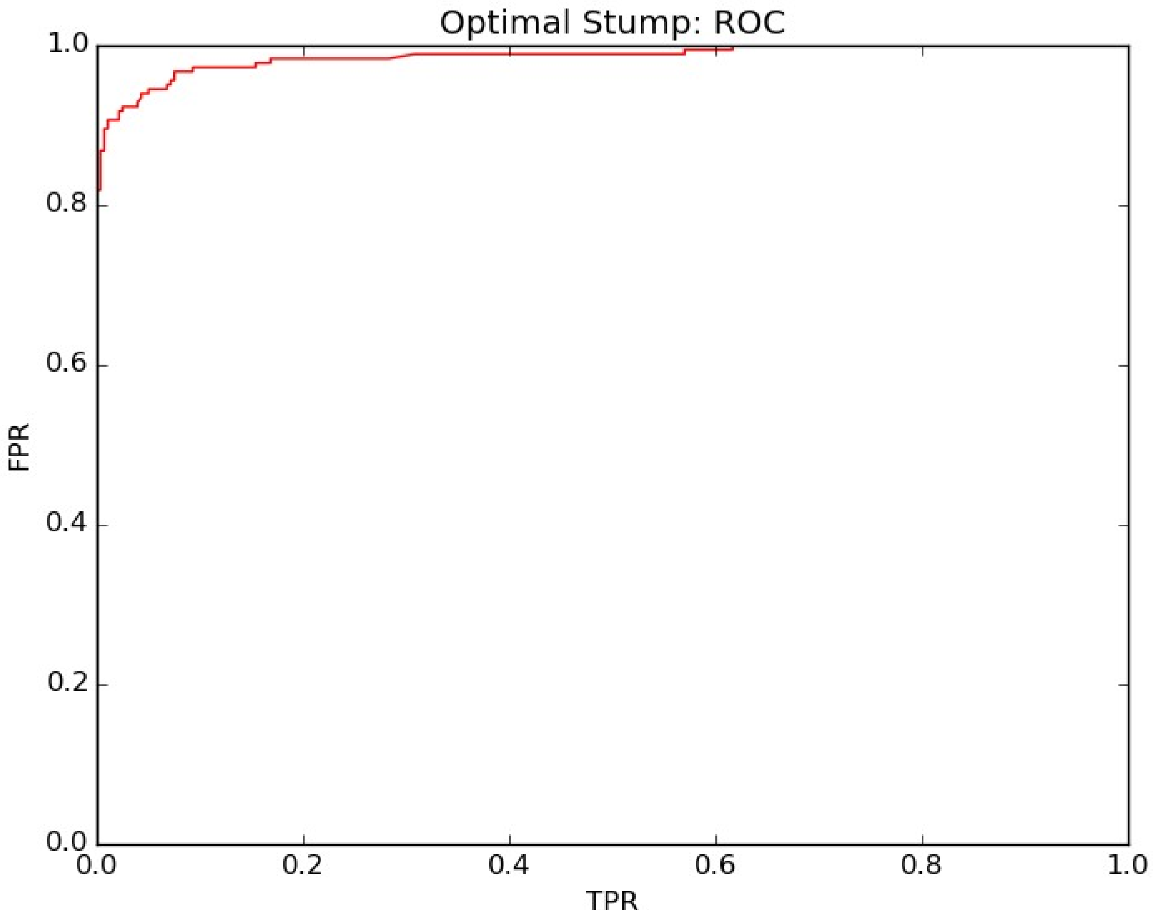
Luke Jaffe
CS6140 Machine Learning
11/07/15

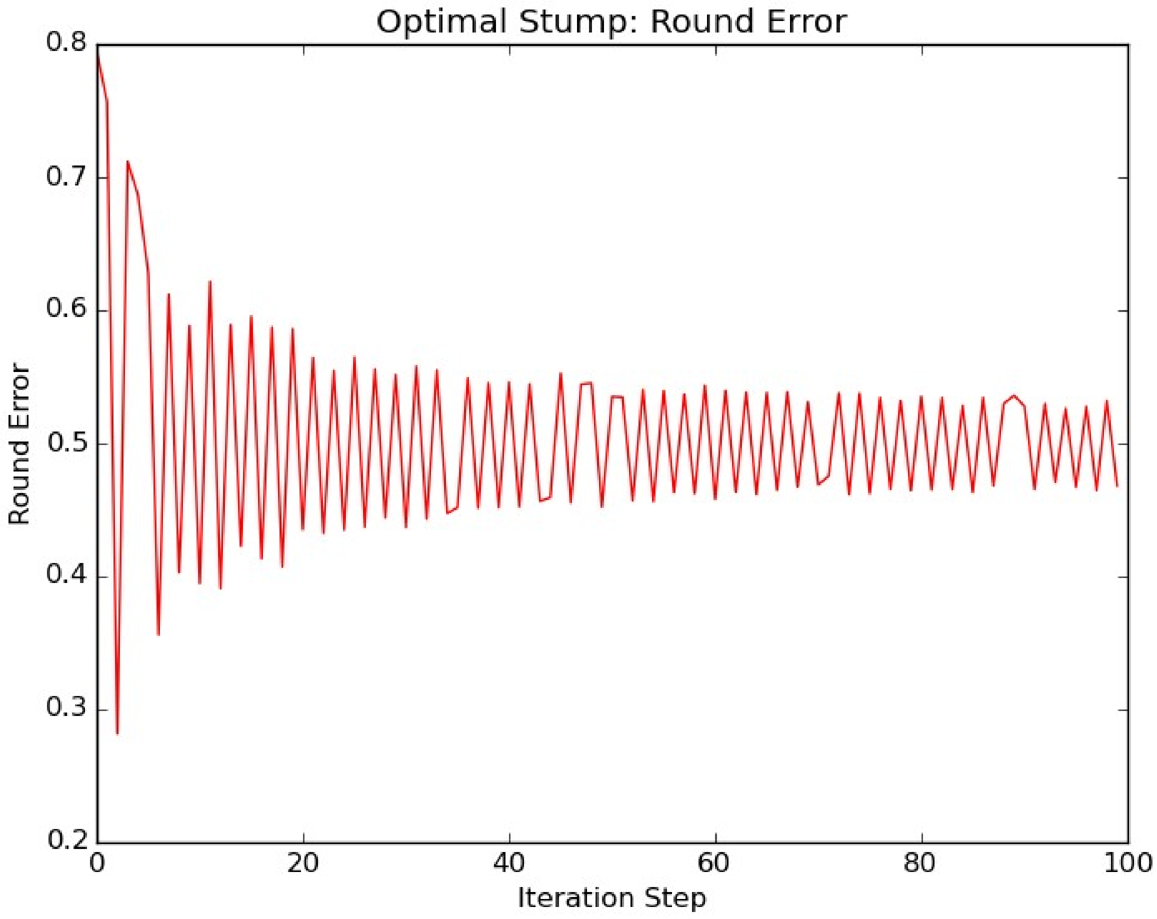
HW4 – Boosting and Bagging

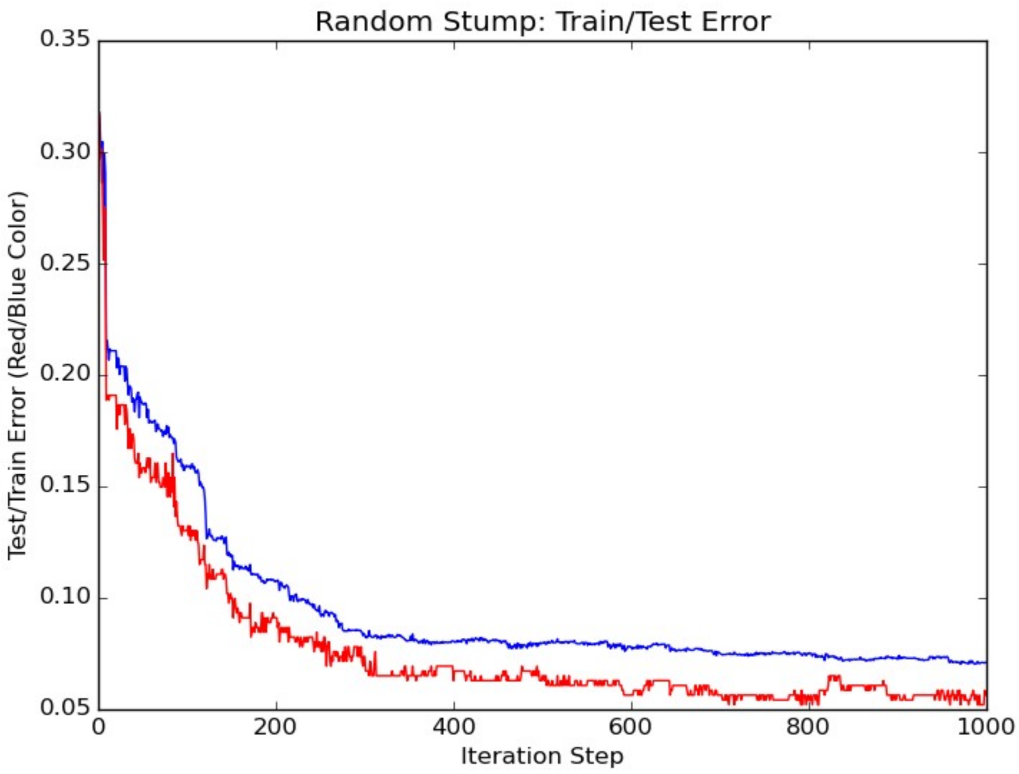
Problem 1. Adaboost Code

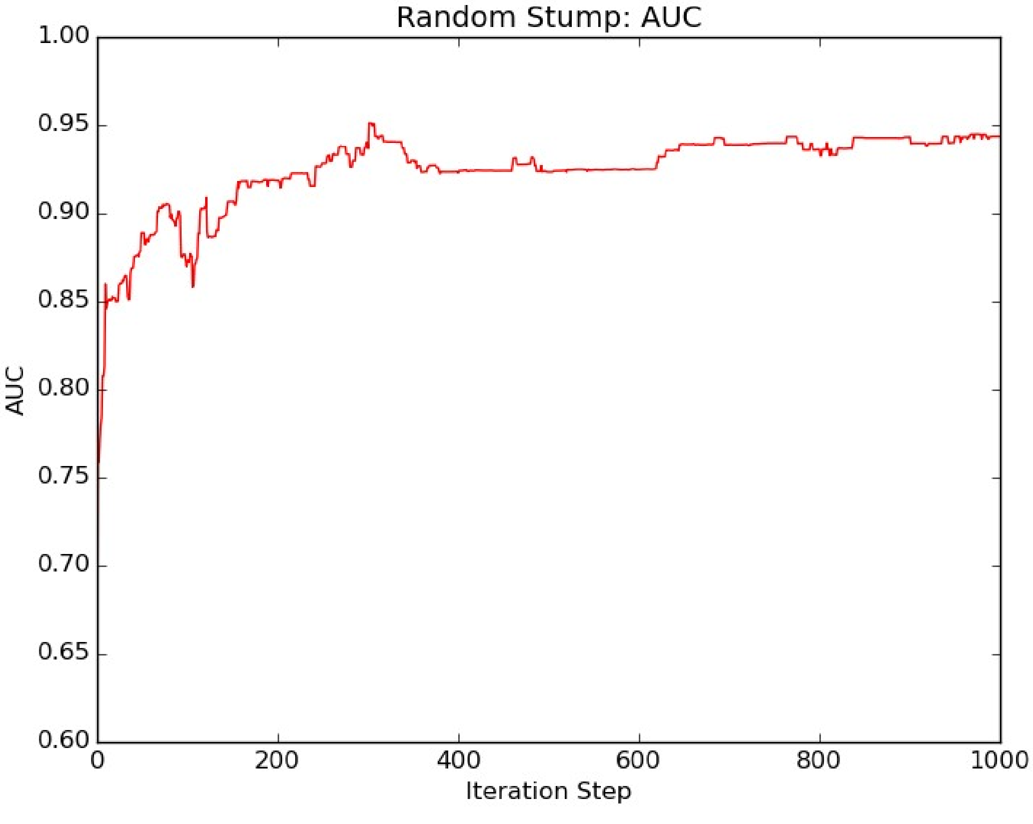
Usage: python adaboost.py

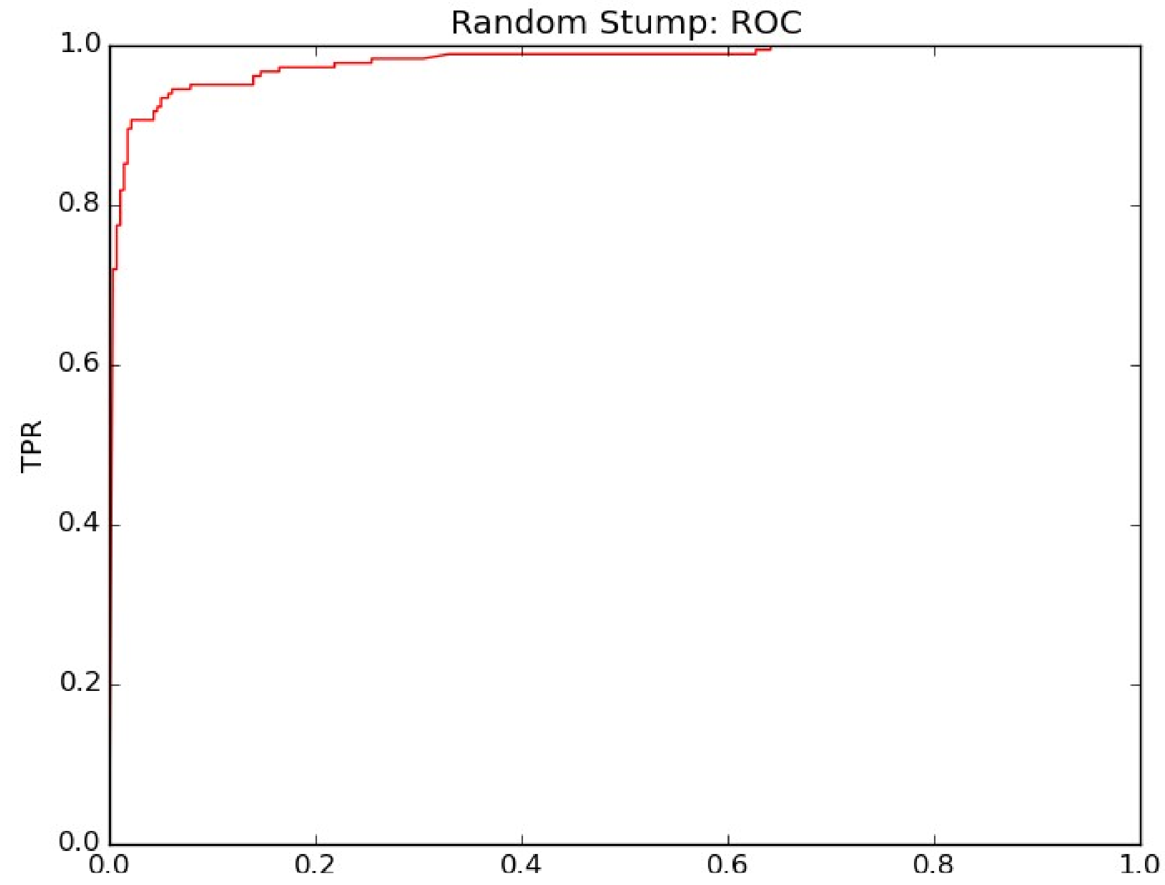


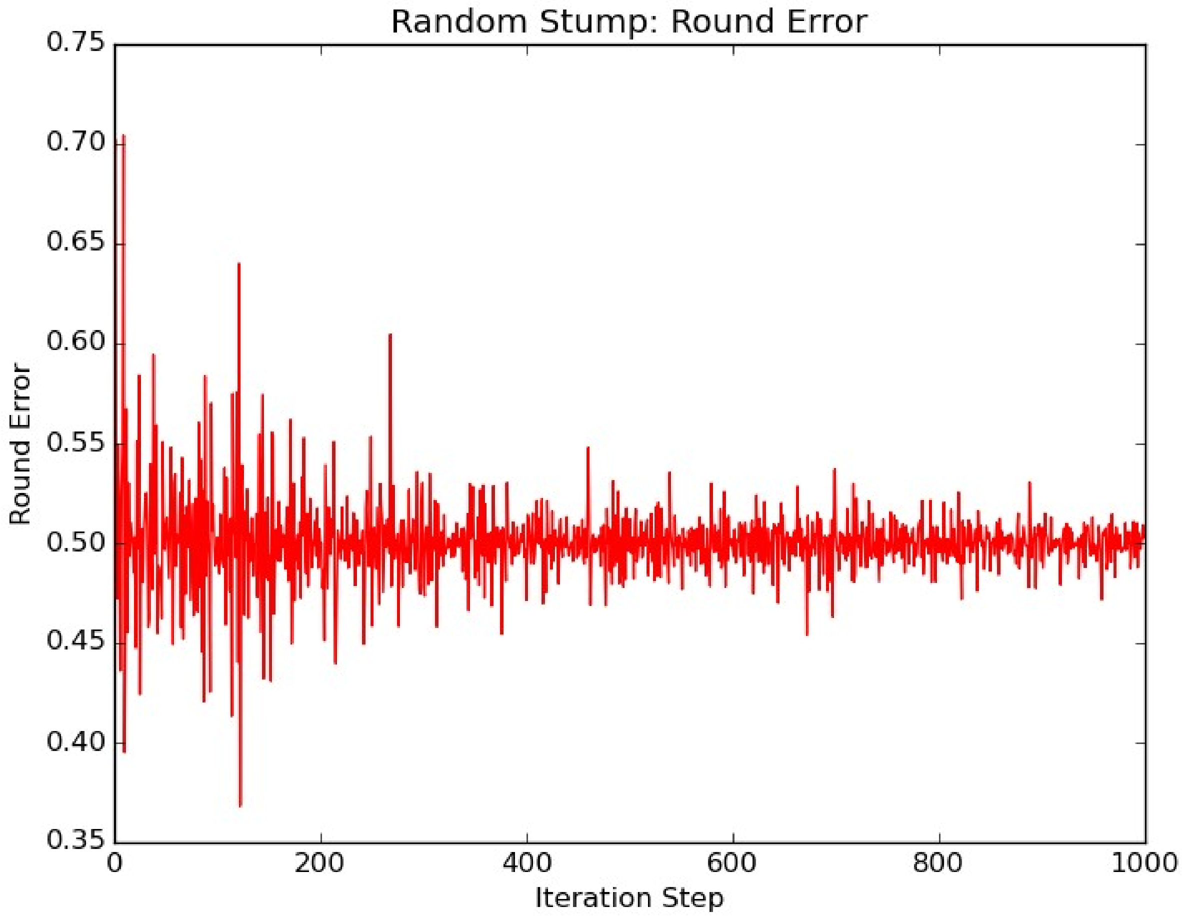

Optimal Stump: ROC
FPR
TPR
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

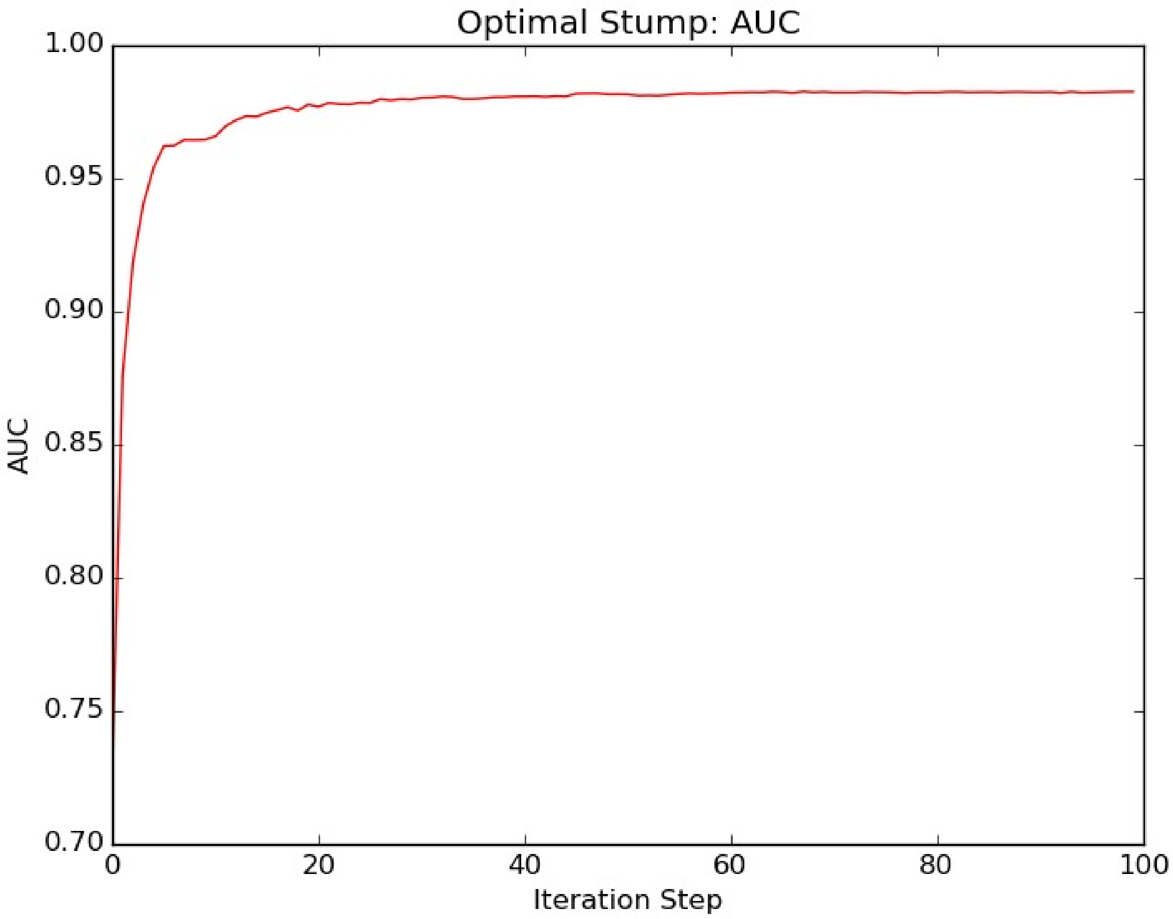

Optimal Stump: Round Error
Round Error
Iteration Step
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
20
40
60
80
100


Random Stump: Train/Test Error
Test/Train Error (Red/Blue Color)
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
200
400
600
800
1000
Iteration Step


Random Stump: AUC
AUC
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
200
400
600
800
1000
Iteration Step


Random Stump: ROC
TPR
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Random Stump: Round Error
Round Error
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
200
400
600
800
1000
Iteration Step



Problem 2. Adaboost on UCI Datasets

a)
Usage: python kboost.py -d {'crx', 'vote'}

Vote:
Average test_acc: 0.951638477801

CRX:
Average test_acc: 0.859420289855

b)
Usage: python cboost.py -d {'crx', 'vote'}

Vote:
c%=5%, train error: 0.00 test error: 0.14 AUC: 0.87
c%=10%, train error: 0.00 test error: 0.14 AUC: 0.86
c%=15%, train error: 0.00 test error: 0.11 AUC: 0.88
c%=20%, train error: 0.00 test error: 0.07 AUC: 0.93
c%=30%, train error: 0.01 test error: 0.06 AUC: 0.96

c%=50%, train error: 0.04 test error: 0.04 AUC: 0.97
c%=80%, train error: 0.04 test error: 0.04 AUC: 0.98

CRX:
c%=5%, train error: 0.00 test error: 0.41 AUC: 0.61
c%=10%, train error: 0.00 test error: 0.26 AUC: 0.82
c%=15%, train error: 0.00 test error: 0.19 AUC: 0.87
c%=20%, train error: 0.00 test error: 0.19 AUC: 0.86
c%=30%, train error: 0.02 test error: 0.20 AUC: 0.86
c%=50%, train error: 0.06 test error: 0.14 AUC: 0.90
c%=80%, train error: 0.09 test error: 0.11 AUC: 0.95

Problem 3. Active Learning
Usage: python active.py

Starting with 5% of the data, and increasing by 2% per round (final result):
Train error: 0.0767184035477
Test error: 0.0478260869565

Problem 4. Error Correcting Output Codes
Usage: python ecoc.py

100 rounds training on ~45% of dataset (full testing set used)

Training Accuracy: 0.7578
Testing Accuracy: 0.714816781731

Problem 5. VC Dimension

a) Unions of Two Rectangles

If the VC-dim of a single rectangle is four, we know the vc-dim of the unions of two rectangles is at least eight, since we could have two separate sets of four points, each shattered by a single rectangle. The classifier must also be able to shatter at least 9 points: If 9 points are positioned around a circle, then the minimum label is comprised of at most 4 points. Each rectangle can rope any 2 points, so this would be satisified. However, take a circle with 10 points: If the labels alternate +,-,+,-, then there is no way to position the two rectangles to shatter the points. Therefore, the VC-dim=9.

b) Circles

It is clear that the VC-dim is at least three. For any three points, there are either two or three points labelled as the majority. In the case of two being labelled as, say, +1, the circle can be drawn around the -1. If The majority has all three points, the circle can of course be drawn around all three. To show that VC-dim is less than four, let us imagine a scenario with four points position in a square. If the labels alternate, then no circle can capture opposite corners without capturing the other points as well. Therefore, VC-dim=3.

c) Triangles

Take seven points on the edge of a circle. Since the minority label always comprises three or less points, a triangle can clearly be drawn with these points. However, when we add another point, the minority label comprises up to four points. In this case, the triangle cannot satisfy all possible labellings. In fact, there is no set of eight points which can be shattered by the triangle classifer, so its VC-dimension=7.

d) D-Dimensional Sphere

Actually, this is quite similar to the circle case. Each added dimension only increases the VC-dim of the classifier by one. Therefore, the VC-dim of the D-Dimensional sphereis VC-dim=D+1.

Problem 6. Bagging

Usage: python tree.py

```
Testing accuracy (no bagging): 0.900217
Testing accuracy (bagging): 0.913232
```

Problem 7. Gradient Boosted Trees for Regression

Usage: python rtree.py

With 10 rounds of boosting, and tree depth of 5:

```
Training MSE (boosting): [ 2.80541994]
Testing MSE (boosting): [ 17.99761449]
```