**Problem 1:**

**1.**

VC dimension for problem 3-d boxes is **2**.

**Prove:**

For any 2 points: When the 2 points labeled + +, there can be an axis aligned 3-d boxes contained the line segment. When the 2 points labeled - -, there can be an axis aligned 3-d boxes outside the line segment. When the 2 points labeled + -, there can be an axis aligned 3-d boxes contained the + point.

For any 3 points, let the three points labeled + + -, the two points formed a line segment not aligned with any axis, then the line segment can form a box that takes the line segment as the diagonal line. Let the third point labeled – inside the box. Because the box we assumed is the smallest one, then there is no box can separate the three points.

Use formula M >= - (1/ε) ln(σ/|H|) = -(1/0.2) ln (0.05/2) > 18.4, So 19 samples would be enough to guarantee an optimal learning algorithm for attaining an accuracy of .8 with probability at least .95.

VC dimension of R1 is 2, VC dimension of R2 should be 2, of R3 should be 2, as the VC dimension keep constant when dimension increase. So the VC dimension of Rd should be 2. Then the number of enough samples will be **19**.

**2.**

**4**

The point inside rectangle be +1, let the point outside be -1, then

f(x) = sign(a1 + a2) if point in both h1(x) and h2(x)

f(x) = sign(- a1 - a2) if point not in h1(x) and h2(x)

f(x) = sign(a1 - a2) if point just in h1(x)

f(x) = sign(-a1 + a2) if point just in h2(x)

So we just consider two box in the R2 dimension.

Let the count of + be C.

For any 4 points, C<=2 or C==4, there clearly are two box contained them. When C == 3, it can constructe a triangle. Traversing all situations, either – outside the triangle or inside the triangle, the two boxes can contain 2 points and 1 points satisfying the requirements.

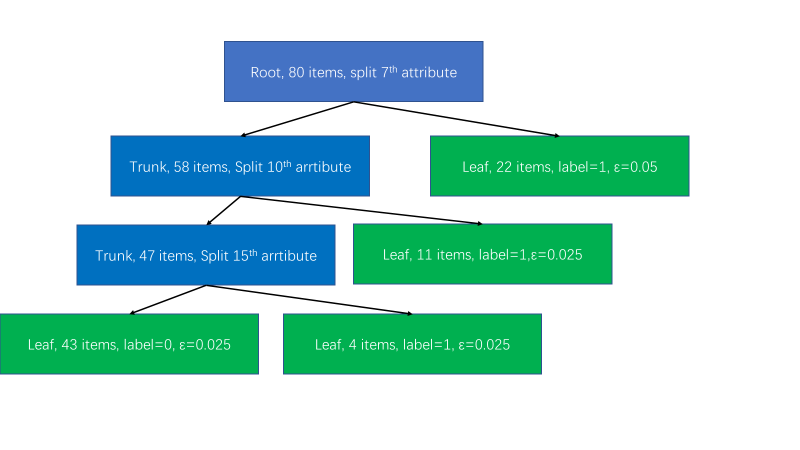
For 5 points, when C == 3, the + construct a triangle. Let two points – inside the triangle, there are situation two box cannot separate the three + points.

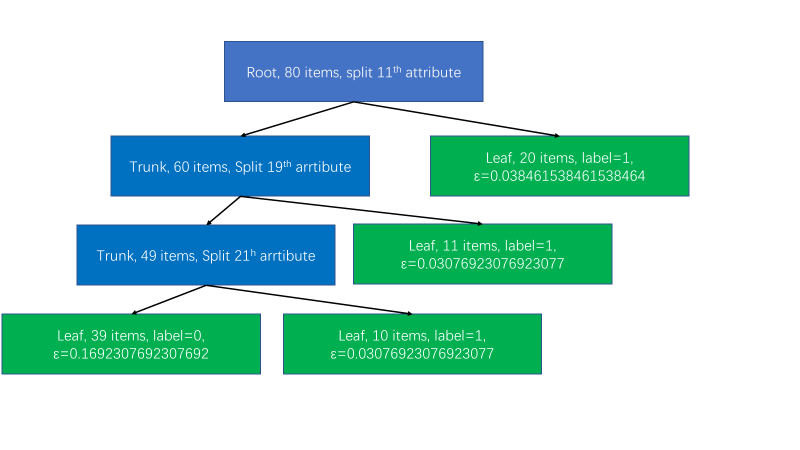
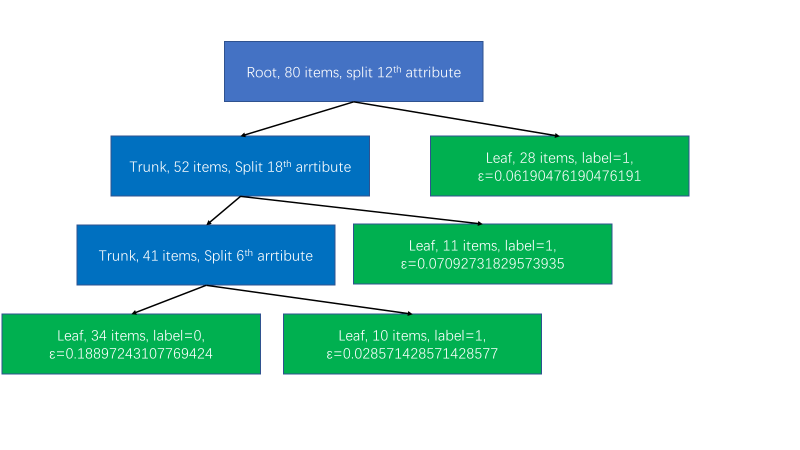
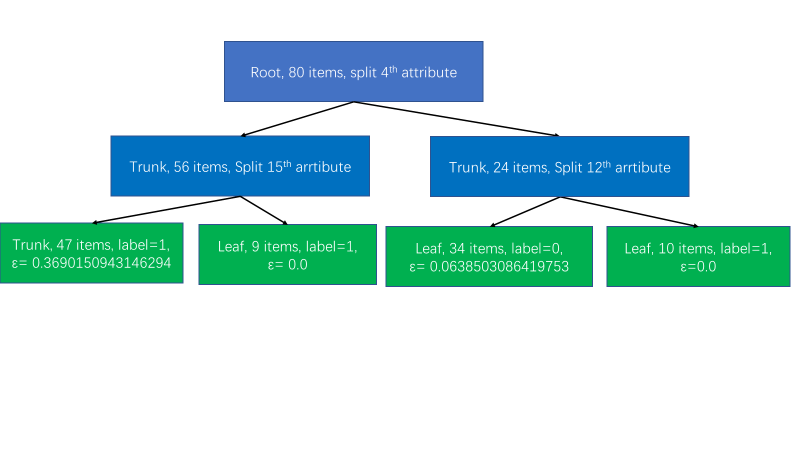
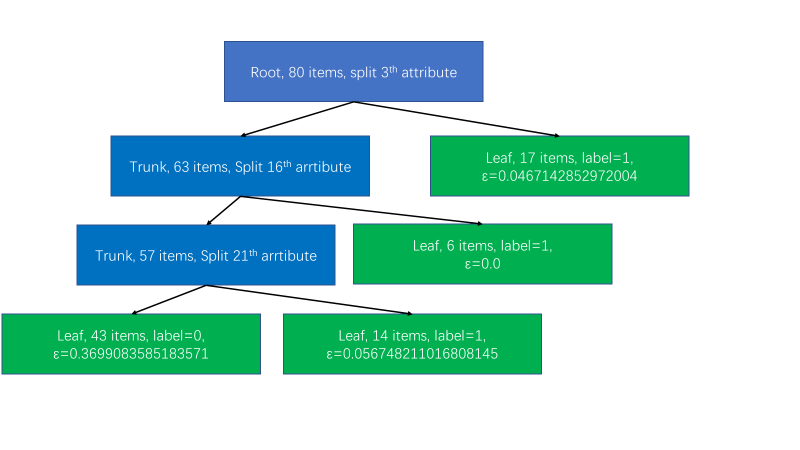
So the VC dimension should be 4.

**Problem 2:**

1. **a)**

Please take code file boosting2.py for reference.

****

**** ****  

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | ε | α | |
| 1 | 0.1875 | | | 0.7331685343967135 | |
| 2 | 0.2692307692307692 | | | 0.4992644150555637 | |
| 3 | 0.35037593984962406 | | | 0.30869356733920494 | |
| 4 | 0.4328654029566047 | | | 0.1350849140543451 | |
| 5 | 0.4733708548323657 | | | 0.0533087309521715 | |

1. **b)**

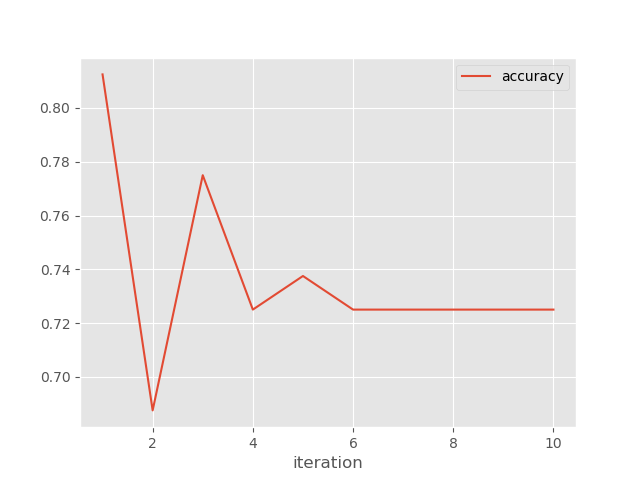
Please take code file boosting2\_2.py for reference.

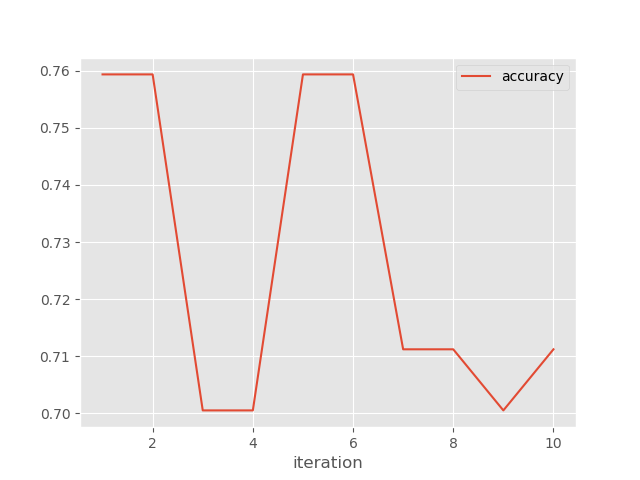
The first chart is accuracy of training data of each round of 10 rounds.

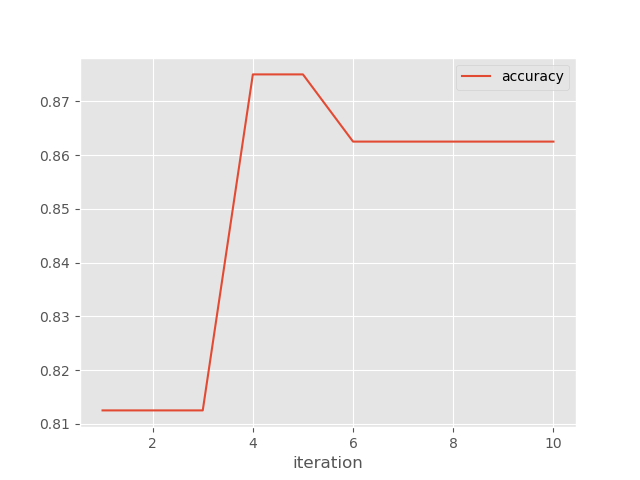
The second chart is accuracy of test data of each round of 10 rounds.

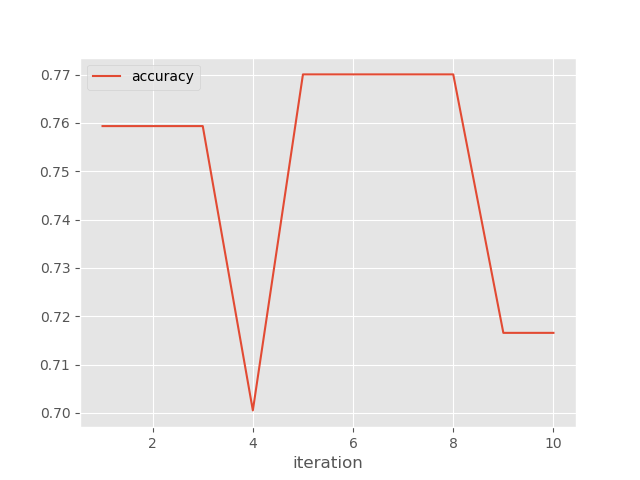
The third chrt is training data accuracy of adaboost result for 1,2,3…10 rounds.

The fourth chrt is test data accuracy of adaboost result for 1,2,3…10 rounds.









1. **a)**

Please take code file boosting3.py for reference.

optimal value of α: [13.766577515296945, 8.039148758873553, 6.320086040558312, 15.224324283796767, 14.914490639855657, 15.328822078532962, 2.54002021684273, 3.311180539472315, 17.782642236634036, 15.83549143311343, 7.008795130814915, 6.108299198880173, 3.474847334880833, 19.250400719253015, 8.539093585195877, 6.47482783487613, 16.878758746244102, 18.534879170980236, 18.052090566100215, 1.372693561474598, 7.485566909152999, 1.350853997904889]

exponential loss: 8.5406283323533525e+99

**b)**

Please take code file boosting3\_2.py for reference.

accuracy: 0.31016042780748665

**c)**

Please take code file boosting3\_3.py for reference.

accuracy: 0.6149732620320856

From accuracy we can see adaBoost does much better than gradient descent.

The α for adaBoost: [4.84700279e-01 3.20926943e-01 1.31007541e-01 3.83774746e-02 9.83889369e-03 2.42693018e-03 5.92774094e-04 1.44433114e-04

3.51711745e-05 8.56336079e-06 2.08490532e-06 5.07603623e-07

1.23583982e-07 3.00884234e-08 7.32548905e-09 1.78350290e-09

4.34221214e-10 1.05717879e-10 2.57385224e-11 6.26654284e-12]

The α in adaBoost keep decrease until approach 0. It means our learner can converge to a stable learner.

The α in gradient descent does not follow such a rule. The learner hardly improves as the iteration increase.

**d)**

Please take code file boosting3\_4.py for reference.

accuracy: 0.6149732620320856

The bagging method does better than gradient descent. The bagging method performance same as adaBoost.

**e)**

AdaBoost. From above we can see the adaBoost give the best accuracy performance. It always has a high accuracy in training dataset. When it has low bias, it may also introduce higher variance. In addition, adaBoost has the standard formula to follow. So, I prefer adaBoost for this dataset.