|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Y | X1 | X2 | X3 | X4 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |

1.3: likely prove this with code. Ask for code to prove that it’s not linearly seperable. – perceptron

Question 2: hope is lost. Gotta get back… back to the past

Question 3:

Notes: C is a collection of functions, each function of C1 (each function) is f\_z(x) where if z = x then its true.

* z, if n = 3 per se, could be 000, 100, 010, 001, 110, 011, 111 101,… (all the possible combinations).
* x is a possible combination. aka 100, 010, etc. If z = x, then f\_z(x) =1, 0 if otherwise.

So the size of the concept class is 2^n.

* example: n=2. There are 2^2 possible combinations of 0 and 1. 00, 10, 01, and 11. That is the size of the concept class C.

b. A mistake bound algorithm that makes no more than one mistake on any sequence of examples presented to it.

- so, for instance, lets use n=2 again. That means our possible z values are 00, 10, 01, and 11.

- a sequence of examples could be 11, 00, 11, 01.

- we need to only make 1 mistake on that sequence, other than that one mistake, the algorithm must classify x correctly.

- Likely idea: use the elimination algorithm. Except… that algorithm is for conjuctions…. This isn’t conjunctions. Whether or not f\_z(x) returns 1 depends on the value/ sequence of x… which, isn’t that random? (\*\*\* note: further down, we realize that this actually kind is about conjuctions)

- Also remember how it gets checked in the first place. Say x is 01 and its value is 0. If we match that against 01 in C, we’d get 1, which is incorrect.

- From the problem description, the goal is to come up with an algorithm that will learn any function f in C1. I’m assuming this means that there’s only one value of x that will give a 1 in any set of examples.

- so for example, if the function we’re trying to find is 01. Then a sequence could look like this: 00 - 0, 11 - 0, 01 – 1.

- now, how would that work in an algorithm that uses C1 to define/hypothesize outputs of x? I don’t think elimination would work, because that only considers ‘monotone’ conjunctions I think, that is, there can’t be any ‘not’s in the function, only ‘and’s. CON won’t work either, it can make n-1 mistakes.

- however, as an idea, CON does loop through every algorithm that is still in the set C. So we can probs do that too.

- its literally just the CON algorithm. It won’t make more than 1 mistake, because once you make a mistake, the algorithm will eliminate all other functions from C1, which will leave you with the right one.

Algorithm: CON

* check:
  + of the available functions in C1, choose 1 randomly to predict the value of x.
* update:
  + If correct, remove all functions but the one just used.
  + If incorrect, search the current functions for the z that matches x. Eliminate all functions except the one where the z for that function matches the current x.

For instance, if x = 11, and f(x) = 0, and we chose z=01 for our function f\_z(x), then… that’s actually right. f(x) = 0 and f\_z(x) = 0.

If we got it wrong though there’s two possibilities:

* Predicted: 0, Actual, 1.
  + Actual function: 01. Given x: 01. Chosen z: 11.
  + If this is the case, remove all but the z that matches x.
* Predicted: 1, Actual: 0.
  + Actual function: 01. Given: 00. Chosen z: 00
  + How does this tell use anything?

2.

C2 consists of n functions. Fi(x) = xi. Which means fi returns the value of the ith feature of x.

Example: x = 011001. For i=3, fi(x) = 1.

\*\* CON algorithm. There exists a function f in C that we want to learn / is the RIGHT one. (u can look at the video but…) You randomly choose a function from C, and if that functions prediction was WRONG, then remove all functions that got that same result.

* SO for a set of functions C = {f0,f1, f2 ,f3}, say the true function (we don’t really know this in the algorithm yet) is f3. If you run an example x and your randomly chosen f gives 1 but the actual output of x was 0, then remove all functions in C that got an output of 1.

1. O(N-1)

Well, say n= 4. Then there are 4 possible functions, 1 is right. We could encounter all 3 incorrect functions and remove them first before we get to the right one.

1. O(Log(C)) mistakes.

Ya know, because. Of the N functions, we will eliminate at least half of them on each iteration (where there is potential to make a mistake). So it will take at most Log(C) mistakes to reach the correct function. For instance, if n=8, then

* Iteration 1: |C| = 8, |C\_+1| = |C| \* ½
* Iteration 2: |C| = 4, |C\_+1| = |C| \* ½
* Iteration 3: |C| = 2, |C\_+1| = |C| \* ½
* Iteration 4: |C| = 1, |C\_+1| = |C| \* ½

Log2(8) = 3. Aka there were 3 mistakes made before we reached |C| = 1, where we have found our true function.

===== Currently, 1.3, 2.1, 2.2, and 3.1.b are unknown. ===== that sux.

<<<On to perceptron>>>