# Problem 3.41

1. The low-degree algorithm is this: such that , use to estimate up to accuracy with confidence and then output . samples each time, where and . So, .We can just use the same examples each time (call it the batch ).
2. The final hypothesis is

where

Because is always positive, it has no effect on the sign of the expression and thus can be eliminated.

I don’t really see how each example’s weight only depends on its Hamming distance from because it also depends on , as shown in this expression. It does *include* the Hamming distance, however, because when , , where is the Hamming distance between and .

# Problem 4.4

1. A DNF of width can be computed by a depth decision tree. Using Exercise 3.30, it becomes , as because our function can only output either or . So, is . Then using proposition 4.9, we get that our function is computable by a DNF of width due to the law of logs, which then is because is a constant. We prove the claim by putting these two together using the fact that .
2. We learned in class that if function is computable by a size DNF, then is -concentrated to degree . In this case, . The Low-Degree algorithm says for all in a concept class such that is concentrated up to degree , can be learned in time with error . In our case, . We know that by big Omega notation, , and is constant for a fixed function , so becomes . Similarly, is , which becomes . Together, is learnable in time with error .

# Problem 4.19

1. By “doesn’t falsify ”, I’m taking this to mean “making sure is not constantly ”. We already know is not constant, which means there’s no restricted variables in because otherwise, will just always be false, as it’s a conjunction. To make sure is not always false, all you have to do is, if it’s in set ; if it’s , set .
2. For a restriction to be bad, at least one has to be neither constantly or , and for that to be “bad” (i.e. neither constantly or ), at least one variable in it has to be free. A “bad” could have anywhere from 1 to free variables, so using what we did in part (a), literal could appear as one of one to one of free variables in a clause, so the corresponding could appear a maximum of times
3. S
4. S