# Assignment 1

## PAC Learning

1. Let F be a learning framework describing conjunction of literals.

F is PAC learnable because there exist an algorithm that can learn it in polynomial time. In this algorith h starts such that h contains all literals, positive and negative. h only gets updated when the algorithm receives a positive example e. When it does, it removes all literals from h that e did not fulfil. This process never removes literals from h that are in t, since e is only positive when e fulfils all literals in t. This means that that h will always classify an example as negative if t does. It can only misclassified positive examples.

When h misclassify an example e, h has at least one bad literal z, that e does not fulfil. The probability of picking such an example given a literal z is Dk(e in m(t) | e does not fulfil z ). The probability of picking an example not fulfilling any literal z of h is:   
sum z Dk(e in m(t) | e does not fulfil z).   
error(h, t, D) = sum z Dk(e in m(t) | e does not fulfil z).  
So long as Dk(e in m(t) | e does not fulfil z) for all z’s is less or equal to epsilon/2n, then the total probability is not gone be bigger then epsilon. Because of this the problematic literals z are those where Dk(e in m(t) | e does not fulfil z) >= epsilon/2n.

The probability of any specific bad literal z not being detected after one draw is (1 – eps/2n), for m draws it is (1 – eps/2n)^m. The probability for any arbitrary bad z not being detected must therefore be equal or less then 2n(1 – eps/2n)^m. The goal is to pick an m such that 2n(1 – eps/2n)^m <= delta

Using (1+x) < e^x inequalities, can we show that picking m where 2n \* e^(-eps\* m/2n) <= delta, Will satisfice 2n(1 – eps/2n)^m <= delta.

From 2n \* e^(-eps\* m/2n) <= delta we get m >= (2n)/epsilon\* (ln(2n)+ln(1/delta))

2.  
There is a h = t such that for any S errorS(t,h,D) = 0. For any S there is at least on h such that errorS(t,h,D) = 0. Let A be an algorithm that assigns any S a h such that S errorS(t,h,D) = 0. We can ignore any h that does not have any corresponding S where A(S) = h, because they will never be chosen. For every other h can we pick at least on Sh where A(S) = h

A bad hypothesis h is defined as a h such that . If there is no then   
For any fixed bad h the probability of not picking an e such that A will not chose h is

for m picks that is

We then get at most for all possible bad hypotheses.  
Then we need to pick m such that   
Because of inequality we can pick m such that:  
Which can be simplified to   
m is finite as long as |H| is finite.

## Hands on code

Hypotheses and valuations are represented as two sets of variables with the class named Simple\_CNF. One for positive and one for negative. Hypotheses can have positive .

Generations are based on a normal distribution. The number generated from the normal distribution, where the binary representation of that numbers tells with of the variables are positive or negative (goes in positive or negative set).

In this implementation , a static target is used. The examples are labelled using target t, after they are generated such that there are examples.

When learning, the algorithm starts with a hypothesis with all laterals (both pos and neg set are filled). The learning algorithm then does one learning step for each example e. Nothing changes if e is labelled as positive. If e is labelled as negative, then redundant literals are removed from h by using intersection.   
  
The hypotheses are then tested by generating new examples. It is then checked how many of those examples t and h evaluated differently. This was then divided by N to get the expected error.

This prosses was repeated times.