Învățare Automată (Machine Learning)

Bogdan Alexe,

bogdan.alexe@fmi.unibuc.ro

Master Informatică, anul I, 2018-2019, cursul 9

Recap - Computational resources of learning

For learning we need 2 type of resources:

1. *Information = training data = sample complexity*

2. *Computation = runtime = computational complexity*
   - runtime = number of elementary instructions executed - arithmetic operations over real numbers - in an asymptotic sense (with respect to input size) of the algorithm, e.g. *O(n) – where n is the size of the input size*

   - the runtime of a learning algorithm A defined as the maximum of:
     - the time it takes A to output some *h*
     - the time it takes *h* to output a label on any given *x* from $\mathcal{X}$

   - want to have *efficient learning* (give a formal definition in the lecture): polynomial in 1/ε, 1/δ and *n* (some parameter related to the size/complexity of domain/hypothesis class: more complex hypothesis needs more computation time)

Recap - Examples

1. $\mathcal{H}_{\text{conj}}^{\text{d}}$ = class of conjunctions of at most *d* Boolean literals $x_1, \ldots, x_{\text{d}}$
   - in the realizable case (PAC learning) the runtime of an algorithm that finds an ERM hypothesis is $O(m_{\mathcal{H}}(\varepsilon,\delta)$*d), so is polynomial in 1/ε, 1/δ , *d*
   - in the agnostic (unrealizable) case: unless P = NP, there is no algorithm whose running time is polynomial in $m_{\mathcal{H}}(\varepsilon,\delta)$ and *d* that is guaranteed to find an ERM hypothesis for the class of Boolean conjunctions.

2. $\mathcal{H}_{\text{rec}}^{\text{d}}$ = the class of axis aligned rectangles in $\mathbf{R}^{\text{d}}$
   - in the realizable case (PAC learning) the runtime of an algorithm that finds an ERM hypothesis is $O(m_{\mathcal{H}}(\varepsilon,\delta)$*d), so is polynomial in 1/ε, 1/δ , *d* (the algorithm has to find the minimal and the maximal values among the positive instances in the training sequence).
   - in the agnostic case for every fixed dimension *d,* ERM_{H} can be implemented in time which is polynomial in 1/ε, 1/δ , *d* (measures the complexity of the $\mathcal{H}_{\text{rec}}^{\text{d}}$) therefore we have efficient learning
   - however, as a function of *d* the runtime of the algorithm implementing the ERM_{H} presented is exponential in *d.* It can be proved that there is no better algorithm (unless P = NP) than the one proposed.

Today’s lecture: Overview

- Randomised algorithms
- Intractability of learning
- Improper learning

Relation between consistency and PAC learning

- a learning rule is consistent if:
  - input: $\mathcal{H}$ and $S = (x_1, y_1), \ldots, (x_m, y_m)$
  - output: $h \in \mathcal{H}$, h is an ERM hypothesis, i.e. $h(x_i) = y_i$
- if VCdim(H) ≤ d then H is PAC learnable (in the realizable case) by the consistent rule with sample complexity:

$$C_1 \frac{d + \log(1/\delta)}{\epsilon} \leq m_{\mathcal{H}}(\epsilon, \delta) \leq C_2 \frac{d \log(1/\epsilon) + \log(1/\delta)}{\epsilon}$$

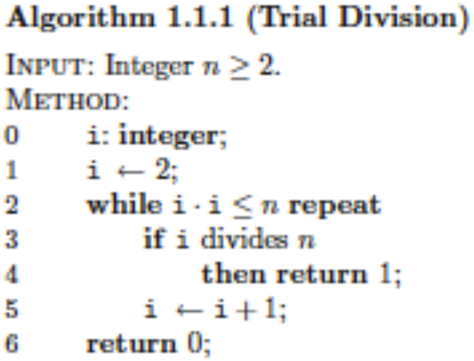
- if d is polynomialy in n and the consistent rule has runtime polynomial in the sample size $m_{\mathcal{H}}(\varepsilon, \delta)$ then we have efficient PAC learning
- so, efficient 'consistent-hypothesis-finder' → efficient PAC learning
- does the converse implication holds?
  - yes, based on randomised algorithms

Randomised algorithms

- a randomised algorithm A is allowed to use random numbers as part of its input
- the output of the algorithm A depends on the input, so it depends on the particular sequence produced by a random number generator
- we can speak of the probability that A has a given outcome
- a randomised algorithm A 'solves' a binary decision problem (YES/NO) if it behaves in the following way:
  - the algorithm always halts and produces an output
  - if the output of A is YES then the answer is 100% right
  - if the output of A is NO then the answer is right with probability ≥ 50%
- randomised algorithm for the problem "number *n* is composite (is not prime)"
  - if the output is YES then we are sure that number *n* is composite (not prime)
  - if the output is NO then with probability ≥ 50% number *n* is not composite (is prime)
- practical usefulness of randomised algorithms: repeating the algorithm several times dramatically increases the likelihood of being correct
  - if I get only NO's in m attempts then with probability of error $\leq 0.5^m$ can say that *n* is not composite (prime)

Randomised algorithms – primality testing

- decide wheather or not a number n is prime or not
- applications in cryptography



Algorithm 1.1.1 (Trial Division)

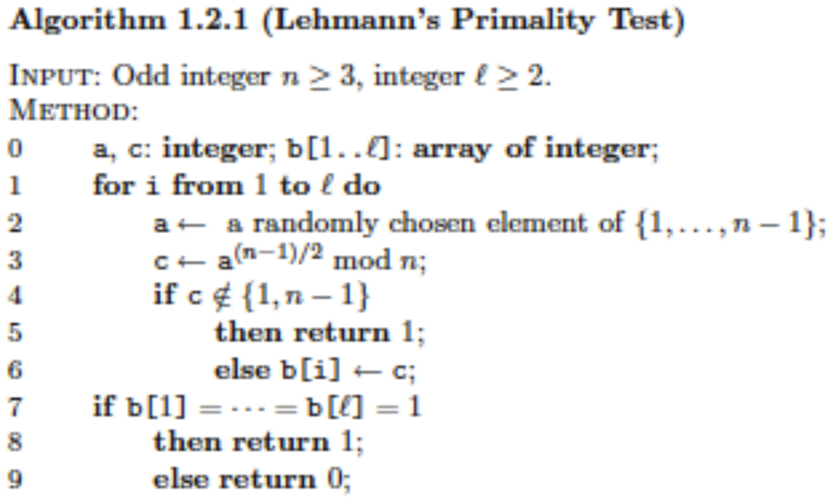
Input: Integer $n \geq 2$.
Method:

```
i: integer;
i ← 2;
while $i \cdot i \leq n$ repeat
    if i divides $n$
        then return 1;
    i ← i + 1;
return 0;
```

- n = 74838457648748954900050464578792347604359487509026452654305481
- n has 62 digits, sqrt(n) has 31 digits
- the basic algorithm takes 10^{13} years to output 0, n is a prime number
- efficient algorithm to test a number being prime?

Randomised algorithms – primality testing

- efficient algorithm to test a number being composite (not prime)?



Algorithm 1.2.1 (Lehmann's Primality Test)

INPUT: Odd integer $n \geq 3$, integer $\ell \geq 2$.
METHOD:

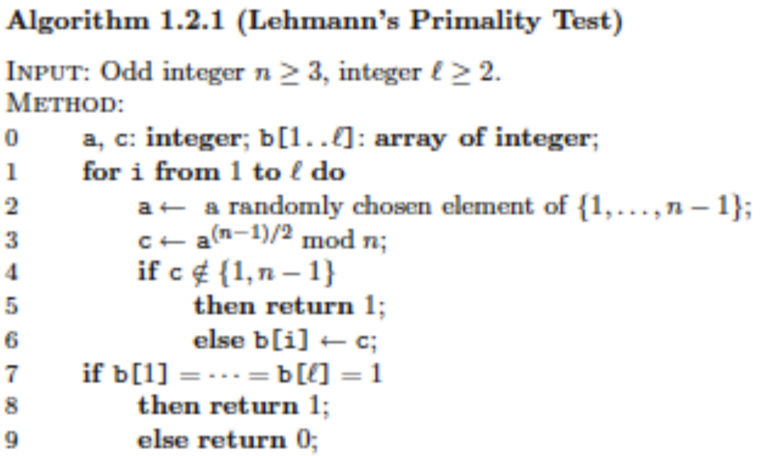
```
0    a, c: integer; b[1..$\ell$]: array of integer;
1    for i from 1 to $\ell$ do
2        a ← a randomly chosen element of $\{1, \ldots, n-1\}$;
3        c ← $a^{(n-1)/2}$ mod $n$;
4        if c $\notin \{1, n-1\}$
5            then return 1;
6            else b[i] ← c;
7    if b[1] = ··· = b[$\ell$] = 1
8        then return 1;
9        else return 0;
```

- the algorithm returns 1 = it takes the decision that the number is composite (not prime)
- the algorithm returns 0 = it takes the decision that the number is not composite (prime)
- repeat for l times lines 2-6
  - take a random number a in {1,…, n-1} and compute c = $\mathbf{a^{(n-1)/2}}$ **mod n**
  - if n is prime then c should be 1 or n-1 (50% - 50% if a is random)
  - n = 7: 1^3 mod 7 = 1, 2^3 mod 7 = 1, 3^3 mod 7 = 6, 4^3 mod 7 = 1, 5^3 mod 7 = 6, 6^3 mod 7 = 6
  - line 5: return 1 if c is not 1 or n-1, so we are sure that n is composite (not prime)
  - if among all c there is at least one = n-1 return 0 (not composite), otherwise return 1 (composite)
  - could have errors

Randomised algorithms – primality testing

- probability of making an error?

- case 1: n is prime (the desired output is 0)
  - 50% chance that c is 1
  - 50% chance that c is n-1
  - error if all c = 1 with probability 2^{-l}



Algorithm 1.2.1 (Lehmann's Primality Test)

```
INPUT: Odd integer $n \geq 3$, integer $\ell \geq 2$.
METHOD:
0    a, c: integer; b[1..$\ell$]: array of integer;
1    for i from 1 to $\ell$ do
2        a ← a randomly chosen element of $\{1, \ldots, n-1\}$;
3        c ← $a^{(n-1)/2}$ mod $n$;
4        if c $\notin \{1, n-1\}$
5            then return 1;
6            else b[i] ← c;
7    if b[1] = ⋯ = b[$\ell$] = 1
8        then return 1;
9        else return 0;
```

- case 2: n is composite (the desired output is 1)
  - make an error if all c are 1 and n-1
  - it can be shown that if there exist some number a in {1, . . ., n − 1} that satisfies $a^{(n-1)/2}$ mod n = n − 1, then more than half of the elements in {1, . . . , n − 1} satisfy $a^{(n-1)/2}$ mod n ∉{1, n − 1}.
  - this means that the probability that the loop in lines 1–6 runs for l rounds is no more than 2^{-l}
  - the probability that output 0 is produced cannot be larger than 2^{-l}

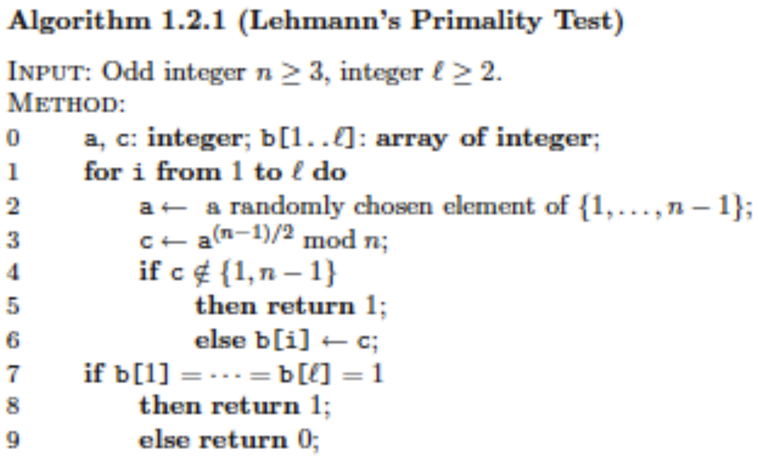
Randomised algorithms – primality testing

- computational complexity?

- do rapid exponentiation in line 3
  - 2 $\log_2 n$ multiplication and divisions of numbers less than n^2

- represent a number n on $\log_{10}(n)$ digits

- overall we have $O((\log_{10} n)^3)$ – suitable on modern computers



Algorithm 1.2.1 (Lehmann's Primality Test)

INPUT: Odd integer $n \geq 3$, integer $\ell \geq 2$.
METHOD:

```
0    a, c: integer; b[1..ℓ]: array of integer;
1    for i from 1 to ℓ do
2        a ← a randomly chosen element of $\{1, \ldots, n-1\}$;
3        c ← $a^{(n-1)/2}$ mod $n$;
4        if c $\notin \{1, n-1\}$
5            then return 1;
6            else b[i] ← c;
7    if b[1] = ··· = b[ℓ] = 1
8        then return 1;
9        else return 0;
```

P $\subseteq$ RP $\subseteq$ NP

- P = class of decision problems that are quickly (polynomial runtime) solvable

- RP = class of decision problems that are quickly (polynomial runtime) solvable by randomised algorithms

- NP = class of decision problems that are quickly checkable (polynomial time)

We have that:

- P $\subseteq$ RP, as we use the same algorithm with polynomial algorithm to solve a problem from P that does not used the random inputs

- RP $\subseteq$ NP, as the solution of the randomised algorithm can be quickly (in polynomial time) checkable

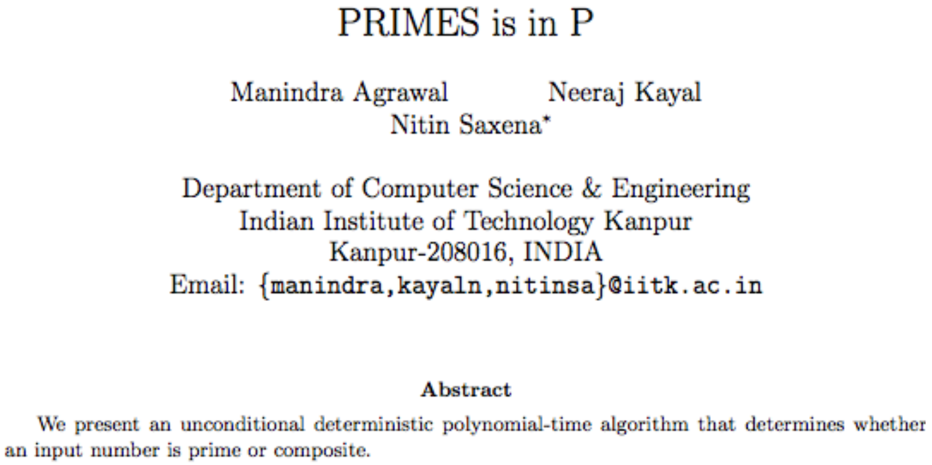
- It is believed that P $\neq$ NP

P = RP ≠ NP ?

- The scientific comunity believes that the following relation holds:

  $$P = RP \neq NP$$

- In 2002, formal proof that there exist a polynomial algorithm to check the primality of a number. Thisindirectly suggests that P = RP.



PRIMES is in P

Manindra Agrawal Neeraj Kayal
Nitin Saxena*

Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
Kanpur-208016, INDIA
Email: {manindra,kayaln,nitinsa}@iitk.ac.in

Abstract

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

Relation between consistency and PAC learning

Theorem

Let $\mathcal{H}$ be a hypothesis class that is efficient PAC learnable. Then, there exists a randomised algorithm which solves the problem of finding a hypothesis in $\mathcal{H}$ consistent with a given training sample, and which has runtime polynomial in *m* (the length of the training sample).

Proof

Let S^* be a training sample, with $|S^*| = m^*$ and let h^* be the function that generates the labels on points in S^*.

Consider the following probability distribution over domain $\mathcal{X}$:

$$\mathcal{D}(x) = \begin{cases} 1/m^*, \text{ if x occurs in } S^* \\ 0, \text{ otherwise} \end{cases}$$

Relation between consistency and PAC learning

$\mathcal{H}$ is efficient PAC learnable, so there exists a learning algorithm A such that: given $\varepsilon = 1/(m^*+1)$, $\delta = ½$, $f = h^*$, *distribution* $\mathcal{D}$ over $\mathcal{X}$ when we run the learning algorithm A on a training set S, consisting of $m \geq m_{\mathcal{H}}(1/(m^*+1), 1/2)$ examples sampled i.i.d. from $\mathcal{D}$ and labeled by h^* the algorithm A returns a hypothesis $h_S \in \mathcal{H}$ such that, with probability at least $1-1/2 = 1/2$ (over the choice of examples), $L_{\mathrm{D},h^*}(h_S) \leq 1/(m^*+1)$.

We want to design a randomised algorithm A* for finding a hypothesis which agrees (training error = 0) with the given training sample S*. We do the following:

- compute $m = m_{\mathcal{H}}(1/(m^*+1), 1/2)$
- construct a training sample S, |S| = m with sample i.i.d from $\mathcal{D}$
- run the given PAC learning algorithm A on S
- check if the hypothesis A(S) = h_S agrees with S*
- if the hypothesis h_S does not agree with S* return NO. Else return h_S. h_S will make 0 error (based on $\mathcal{D}$) so is consistent.

A* will succeed with probability greater than ½. If A is polynomial than also A* is polynomial.

Do we always have efficient PAC learning?

Theorem
Let $\mathcal{H}$ be a hypothesis class that is efficient PAC learnable. Then, there exists a randomised algorithm which solves the problem of finding a hypothesis in $\mathcal{H}$ consistent with a given training sample, and which has runtime polynomial in m (the length of the training sample).

We have now instruments to show that there exist classes that are PAC learnable (in the realizable case) for which the sample complexity is polynomial but the runtime is not polynomial.

Intractability of learning

Learning 3-Term DNF

Consider $\mathcal{H}_{3DNF}{}^{d}$ = class of 3-term disjunctive normal form formulae consisting of hypothesis of the form h: $\{0,1\}^{d} \rightarrow \{0,1\}$,

$$h(\mathbf{x}) = A_1(\mathbf{x}) \vee A_2(\mathbf{x}) \vee A_3(\mathbf{x}),$$

where $A_1(\mathbf{x})$ is a Boolean conjunction (in $\mathcal{H}_{conj}{}^{d}$) with at most d Boolean literals $x_1, \ldots, x_d$.

The output of h($\mathbf{x}$) is 1 if either $A_1(\mathbf{x})$ or $A_2(\mathbf{x})$ or $A_3(\mathbf{x})$ outputs the label 1. If all three conjunctions output the label 0 then h($\mathbf{x}$) = 0.

$|\mathcal{H}_{3DNF}{}^{d}| = 3^{3d} < \infty$, so is PAC learnable with sample complexity 3d log(3/δ)/ε (polynomial in 1/ε, 1/δ, d).

However, from the computational perspective, this learning problem is hard. There is no polynomial time algorithm (unless RP = NP) that properly learns from training data. So ERM_{H} is not efficient.

We will show that if $\mathcal{H}_{3DNF}{}^{d}$ is efficient PAC learnable then the problem of graph 3-coloring problem (which is shown to be NP-complete) is in RP.

Learning 3-Term DNF

Theorem

Unless NP has randomized polynomial time algorithms, there is no proper efficient PAC-learning algorithm for the class $\mathcal{H}_{3DNF}{}^{d}$.

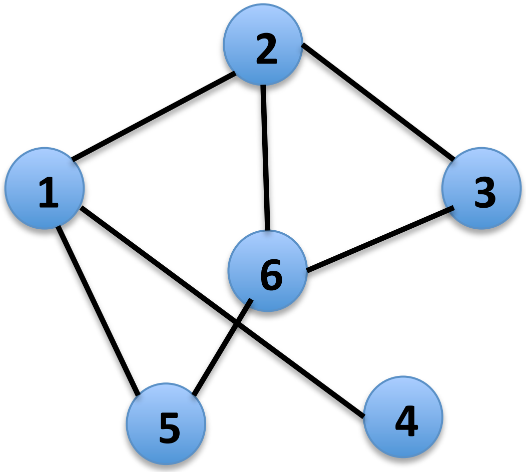
Proof idea

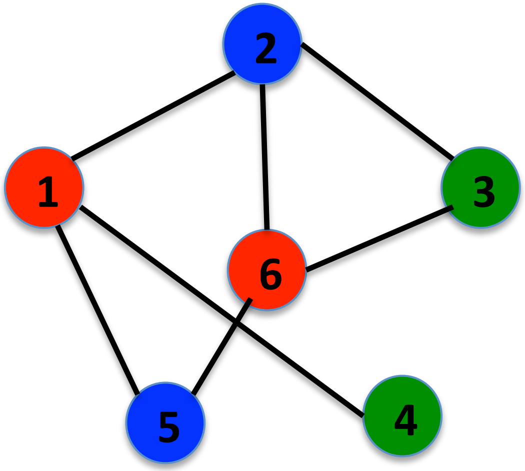
We consider than we can efficiently PAC learn the class $\mathcal{H}_{3DNF}{}^{d}$. From the preceding theorem there exist a randomised algorithms A^* that can decide consistency of 3-term DNF.

We show that the 3-term DNF problem can be reduced to the problem of the graph 3-coloring problem (which is NP-complete). So there exist an algorithm in RP that solves a problem in NP. So RP = NP, which we consider to be false, as we believe that P = RP ≠ NP.

The graph 3-coloring problem

The graph 3-coloring problem.
Given as input an undirected graph G = (V,E) with V = {1, 2, …, n} and edge set E ⊆ V × V, determine if there is an assignment of a color to each element of V such that at most 3 different colors are used, and for every edge (*u*, *v*)∈E, vertex *u* and vertex *v* are assigned different colors.





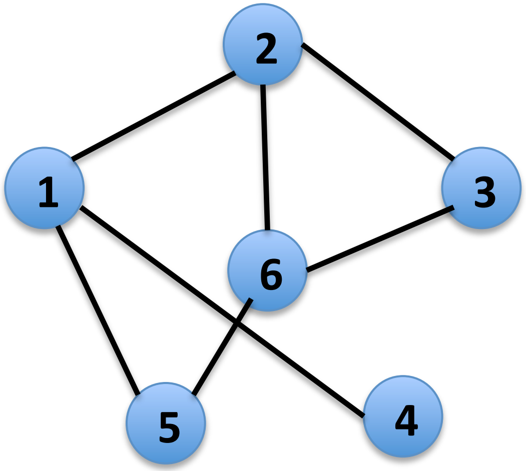
c : V → {R, G, B} is a proper coloring function if for every (u,v) ∈ E we have that c(u) ≠ c(v)

Reduction of graph 3-coloring problem to the 3-term DNF problem

We reduce an instance G = (V, E) to a labeled set S_G of examples (of *n* bits) consisting of |V| positive examples (S_G^+) and |E| negative examples (S_G^-).

Each vertex *i* is encoded as the vector with 0 on position *i* and 1 on all other positions (0-hot encoding) and has label +1: ((1, 1,….1, 0, 1, … 1), +1)

Each edge (*i*, *j*) ∈ E is encoded as the vector with 0 on positions *i* and *j* and 1 on all other positions and has label 0: ((1,….1, 0, 1, ,…1, 0, 1,…,1), 0)

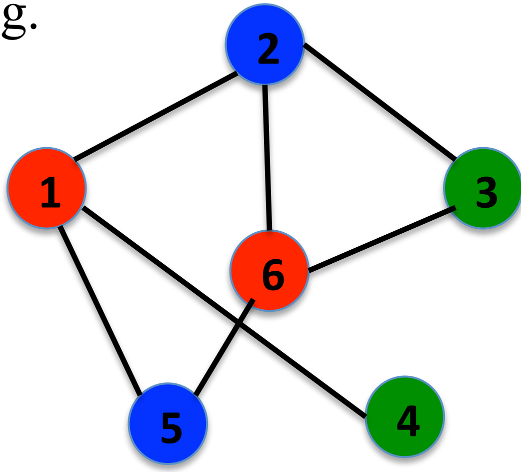


S_G^+	S_G^-
((0, 1, 1, 1, 1, 1), 1)	((0, 0, 1, 1, 1, 1), 0)
((1, 0, 1, 1, 1, 1), 1)	((0, 1, 1, 0, 1, 1), 0)
((1, 1, 0, 1, 1, 1), 1)	((0, 1, 1, 1, 0, 1), 0)
((1, 1, 1, 0, 1, 1), 1)	((1, 0, 0, 1, 1, 1), 0)
((1, 1, 1, 1, 0, 1), 1)	((1, 0, 1, 1, 1, 0), 0)
((1, 1, 1, 1, 1, 0), 1)	((1, 1, 0, 1, 1, 0), 0)
	((1, 1, 1, 1, 0, 0), 0)

Reduction of graph 3-coloring problem to the 3-term DNF problem

The graph G is 3-colorable ⇔ $S_G = S_G^+ \cup S_G^-$ is consistent with some 3-term DNF formula.

Proof "$\Longrightarrow$" Suppose graph G is 3-colorable. Fix such a coloring.



Let R = set of all vertices with color red (= {1,6} in figure)

Let T_R the conjunction of all variables $x_{1,}$ $x_{2,}$ …, x_n whose index does not appear in R. ($T_R = x_2 \wedge x_3 \wedge x_4 \wedge x_5$)

Then, the positive example corresponding to a vertex *i* in R satisfies T_R because x_i does not appear in T_R.

Every negative example corresponding to an edge (i, j) does not satisfy T_R, as either x_i or x_j appear in T_R.

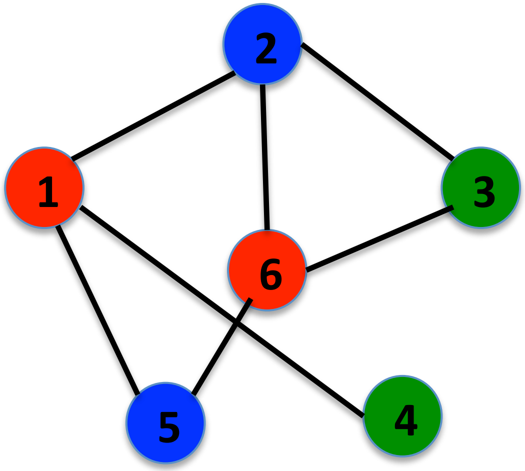
Do the same reasoning for colors green and blue.

Reduction of graph 3-coloring problem to the 3-term DNF problem

The graph G is 3-colorable $\Leftrightarrow$ $S_G = S_G^+ \cup S_G^-$ is consistent with some 3-term DNF formula.

Proof "$\Leftarrow$" Suppose that the formula $T_R \vee T_B \vee T_G$ is consistent with S_G.

Define a coloring of G as follows: the color of vertex *i* is red/green/blue if the positive example corresponding to the vertex *i* satisfies $T_R/T_G/T_B$ (choose random color for ties).



If $i \neq j$ have the same color (let's say red) then:
$T_R(1, ..., 1, 0, 1, \ldots, 1, 1, 1, \ldots, 1) = 1$ (0 on position i)
$T_R(1, ..., 1, 1, 1, \ldots, 1, 0, 1, \ldots, 1) = 1$ (0 on position j)

So literals x_i, $\bar{x}_i$, x_j, $\bar{x}_j$ and $\bar{x}_k$ $(k \neq i, j)$ cannot appear in T_R
So we have that:
$T_R(1, ..., 1, 0, 1, \ldots, 1, 0, 1, \ldots, 1) = 1$. So $(i, j) \notin E$.

Main result

The graph G is 3-colorable $\Leftrightarrow$ $S_G = S_G^+ \cup S_G^-$ is consistent with some 3-term DNF formula.

So, the class $\mathcal{H}_{3DNF}^d$ is not efficient PAC learnable under the assumption that NP-complete problems cannot be solved with high probability by a probabilistic polynomial–time algorithm (technically, under the assumption RP $\neq$ NP).

The technique can be used to generalize the result for $\mathcal{H}_{kDNF}^d$ (k –term DNF formulae), where $k \geq 3$.

"Consistent learning is hard" $\not\Rightarrow$ "learning is hard"

"Consistent learning is hard" $\Longrightarrow$ "*proper* learning is hard"
A proper learning algorithm is a learning algorithm that must output $h \in \mathcal{H}$

Improper learning

Improper learning of 3-term DNFs

Use the distribution rule to obtain:
$(a \wedge b) \vee (c \wedge d) = (a \vee b) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$

Apply for the 3-term DNF formula to obtain a 3-CNF formulae:

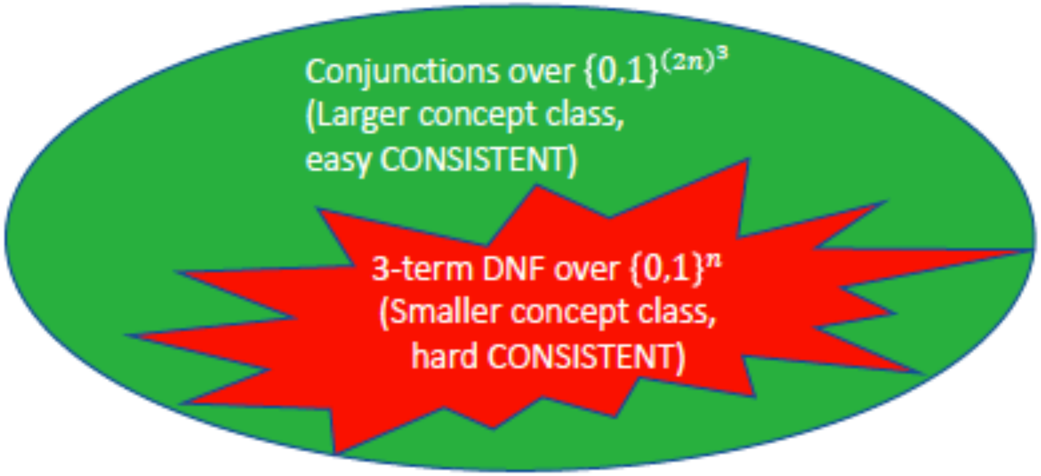
$$A_1 \vee A_2 \vee A_3 = \bigwedge_{u \in A_1, v \in A_2, w \in A_3} (u \vee v \vee w)$$

So we have that a 3-term DNF can be viewed as a conjunction of $(2n)^3$ variables:

$$H_{3DNF}^{n} \subseteq H_{conj}^{(2n)^3} \qquad \left| H_{conj}^{(2n)^3} \right| = 3^{(2n)^3} + 1$$

We can efficiently PAC learn the new class of conjunctions, $H_{conj}^{(2n)^3}$ with sample complexity $(n^3+\log(1/\delta))/\varepsilon$. The overall runtime of this approach is polynomial in $1/\varepsilon$, $1/\delta$, n. Pay polynomially in sample complexity, gain exponentially in computational complexity.

Improper learning



Conjunctions over $\{0,1\}^{(2n)^3}$
(Larger concept class,
easy CONSISTENT)

3-term DNF over $\{0,1\}^n$
(Smaller concept class,
hard CONSISTENT)

	3-term DNF	Conjunctions over $\{0,1\}^{(2n)^3}$
Sample complexity	$O\left(\frac{n+log\frac{1}{\delta}}{\varepsilon}\right)$	$O\left(\frac{n^3+log\frac{1}{\delta}}{\varepsilon}\right)$ ⬆
CONSISTENT Computational-Complexity	NP-Hard	$O\left(n^3 \times \frac{n^3+log\frac{1}{\delta}}{\varepsilon}\right)$ ⬇

Next time – May 10