Definition of CNF

- 3-CNF: CNF with at most 3 literals per clause
- notation: $_{i}^{\wedge}(x_{i_{1}} \vee x_{i_{2}} \vee x_{i_{3}})$
- example: $(a \lor b)(\bar{c} \lor b \lor d)(\bar{a} \lor e)$

Definition of DNF

- 3-term DNF: DNF with 3 terms, each term has as many as 2n literals
- example: $(x_1\bar{x}_2x_3x_4) \vee (x_2\bar{x}_4\bar{x}_6) \vee (x_7x_8\bar{x}_9)$
- k-CNF is PAC learnable
 - o $2^{poly(n)}$ 3-CNF formulas on n literals
 - o reduce 3-CNF to problem of learning conjunctions (already shown):
 - $\bullet \quad T_1 \wedge T_2 \wedge ... \wedge T_n = \bigwedge_{y_1 \in T_1, y_2 \in T_2, ..., y_n \in T_n} (y_1 \wedge y_2 \wedge ... \wedge y_n)$
 - mapping $\binom{2n}{3}$ time is polynomial + conjunction solution is polytime
- k-term DNF ⊆ k-CNF
- 3-term DNF is "probably not" PAC learnable since NP-complete problems cannot be solved efficiently by a randomized algorithm
- Occam Algorithm (already shown):
 - o draw a sample of size $m = poly(n, size(\mathcal{H}))$
 - o return any $h \in \mathcal{H}$ consistent with sample
- $m > \frac{1}{\varepsilon} (\ln(|\mathcal{H}|) + \ln(\frac{1}{\varepsilon}))$

PAC Learning 3-term DNF by 3-CNF

• every 3-term DNF is equivalently representable by 3-CNF, i.e.

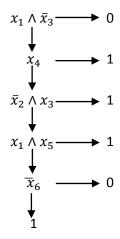
$$\circ \quad T_1 \vee T_2 \vee T_3 = \underset{x \in T_1, y \in T_2, z \in T_3}{\land} (x \vee y \vee z)$$

- converse not true, many 3-CNF cannot be expressed as 3-term DNF because 3-CNF is more expressive than 3-term DNF
- since $\ln(|3 \text{term DNF}|) = \theta(n)$ and $\ln(|3 \text{CNF}|) = \theta\left(\binom{2n}{3}\right) = \theta(n^3)$ can see 3-CNF is bigger than 3-term DNF
- to learn 3-term DNF by 3-CNF use Occam style algorithm:
 - o pick enough samples (find *m* given all other variables)
 - o reduce problem to learning conjunctions to find consistent hypothesis:
 - new set of labeled examples on $\theta(n^3)$ variables, one for each possible 3-CNF clause
- Theorem 1: For any constant $k \ge 2$, the concept class of k-term DNF is efficiently PAC learnable using k-CNF

Definition of *k***-Decision List**

• ordered sequence of if-then-else statements, tested in order

- associated answer corresponds to first satisfied condition
- example:



- formal definition:
 - o k-decision list over variables $x_1, ..., x_n$ is an ordered sequence $L = (c_1, b_1), ..., (c_l, b_l)$
 - where b is a bit value and each c_i is a conjunction of k literals over x_1, \dots, x_n
 - o bit b is the default value, b_i corresponds to condition c_i
 - o for any input $x \in \{0,1\}^n$, L(x) is defined to be bit b_i
 - where j is smallest index satisfying $c_i(x) = 1$, if no index exists then L(x) = b
- denote k-DL as class of concepts that can be represented by a k-decision list

Learning *k***-Decision Lists**

- Lemma 1: $k \text{DNF} \cup k \text{CNF} \subseteq k \text{DL}$
- Proof:
 - concept c can be represented as a k-decision list, C: k-DL
 - \circ ~c can also be represented (compliment the values b_i and b of the decision list representing c)
 - thus this shows k-DNF $\subseteq k$ -DL
 - since k-DNF comes from k-CNF by DeMorgan's Law
 - \circ k-DNF shown as k-DL by choosing an arbitrary ordering to evaluate the terms of the k-DNF and setting all b_i 's to 1 and default value to 0
- Lemma 1 is strict, ∃ functions ∈ k-DL but not by either k-DNF or k-CNF
- **Theorem 2:** For any constant k ≥ 1, the representation class of k-decision lists is efficiently PAC learnable
- Proof:
 - o input sample S, where |S| = m and S is consistent with some k-decision list
 - o build decision list consistent with S as follows:

- find a conjunction $c(\cdot)$ of k literals such that set $S_c = \{x \in S : c(x) = 1\}$ is non-empty and contains only positive or negative examples
- $c(\cdot)$ is a *good* conjunction
- if $S_c = S$ set default bit to 1 if S_c contains only positive examples or to 0 if S_c contains only negative examples
- otherwise add conjunction $c(\cdot)$ with associated bit b as last condition in current hypothesis decision list
- update S to $S S_c$ and iterate until $S = \phi$
- o all initial values given in S are correctly classified by hypothesis k-decision list
- o always consistent because each iteration correctly sets default bit or finds good conjunction
- o any k-decision list on n variables encoded using $\Theta(kn^k \log n)$ bits
 - $M = \# k \text{terms} \le \sum_{i=1}^k {2n \choose i} \approx \theta((2n)^k)$
 - # k terms with values = 2M
 - $\mathcal{H} = \#k DL \le (2M)!$
 - $\log |\mathcal{H}| \approx \log(2M!)$
 - $\log |\mathcal{H}| \approx (2M) \log(2M)$ since $\log(n!) \approx n \log(n)$
 - $\log |\mathcal{H}| \approx (2n)^k k \log(2n)$
- o procedure runs in polynomial time according to m, we have efficient PAC learning