Definition of CNF

- 3-CNF: CNF with at most 3 literals per clause
- notation: $\wedge_i (x_{i_1} \vee x_{i_2} \vee x_{i_3})$
- example: $(a \vee b)(\bar{c} \vee b \vee d)(\bar{a} \vee e)$

Definition of DNF

- 3-term DNF: DNF with 3 terms, each term has as many as 2*n* 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
  - $2^{poly(n)}$ 3-CNF formulas on *n* literals
  - reduce 3-CNF to problem of learning conjunctions (already shown):
    - $T_1 \wedge T_2 \wedge \ldots \wedge T_n = \wedge_{y_1 \in T_1, y_2 \in T_2, \ldots, y_n \in T_n} (y_1 \wedge y_2 \wedge \ldots \wedge y_n)$
    - 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):
  - draw a sample of size $m = poly(n, size(\mathcal{H}))$
  - return any $h \in \mathcal{H}$ consistent with sample
- $m > \frac{1}{\varepsilon}(\ln(|\mathcal{H}|) + \ln\left(\frac{1}{\delta}\right))$

PAC Learning 3-term DNF by 3-CNF

- every 3-term DNF is equivalently representable by 3-CNF, i.e.
  - $T_1 \vee T_2 \vee T_3 = \wedge_{x \in T_1, y \in T_2, z \in T_3} (x \vee y \vee z)$
- 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:
  - pick enough samples (find *m* given all other variables)
  - 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 ≥ 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:

$$\begin{tikzcd}
x_1 \wedge \bar{x}_3 \arrow[r] \arrow[d] & 0 \\
x_4 \arrow[r] \arrow[d] & 1 \\
\bar{x}_2 \wedge x_3 \arrow[r] \arrow[d] & 1 \\
x_1 \wedge x_5 \arrow[r] \arrow[d] & 1 \\
\overline{x}_6 \arrow[r] \arrow[d] & 0 \\
1 &
\end{tikzcd}$$

- formal definition:
  - *k-decision list over variables $x_1, \ldots, x_n$ is an ordered sequence $L = (c_1, b_1), \ldots, (c_l, b_l)$*
    - *where b is a bit value and each c_i is a conjunction of k literals over $x_1, \ldots, x_n$*
  - *bit b is the default value, b_i corresponds to condition c_i*
  - *for any input $x \in \{0,1\}^n$, $L(x)$ is defined to be bit b_j*
    - *where j is smallest index satisfying $c_j(x) = 1$, 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:** $k - \mathrm{DNF} \cup k - \mathrm{CNF} \subseteq k - \mathrm{DL}$
- **Proof:**
  - concept *c* can be represented as a *k*-decision list, *C*: *k*-DL
  - ~*c* can also be represented (compliment the values b_i and *b* of the decision list representing *c*)
  - thus this shows *k*-DNF ⊆ *k*-DL
    - since *k*-DNF comes from *k*-CNF by DeMorgan's Law
  - *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:**
  - input sample *S*, where |*S*| = *m* and *S* is consistent with some *k*-decision list
  - 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 $S = \phi$
  - all initial values given in *S* are correctly classified by hypothesis *k*-decision list
  - always consistent because each iteration correctly sets default bit or finds good conjunction
  - any *k*-decision list on *n* variables encoded using $\Theta(kn^k \log n)$ bits
    - $M = \#\, k - \text{terms} \leq \sum_{i=1}^{k} \binom{2n}{i} \approx \theta\big((2n)^k\big)$
    - $\#\, k - \text{terms with values } = 2M$
    - $\mathcal{H} = \#k - \text{DL} \leq (2M)!$
    - $\log|\mathcal{H}| \approx \log(2M!)$
    - $\log|\mathcal{H}| \approx (2M)\log(2M)$ since $\log(n!) \approx n\log(n)$
    - $\log|\mathcal{H}| \approx (2n)^k k \log(2n)$
  - procedure runs in polynomial time according to *m*, we have efficient PAC learning