CSE 513T Theory of Artificial Intelligence & Machine Learning September 1, 2016

Lecture 2

Lecturer: Brendan Juba *Scribe: Elliot Battle, Editor: Evan Miller*

1 Today's Topics

1. Relative Power of Knowledge Representations
2. Algorithm for Learning Conjunctions
3. PAC-Learning Model (in general)

2 Warm-Up Problem

2.1 Description

Suppose there is a 5% chance that I "give you up" and a 10% chance that I "let you down". Give the best bound you can on the probability that I neither give you up nor let you down.

2.2 Solution

Define the following statements:

1. A = "I give you up"
2. B = "I let you down"

What we are therefore being asked to provide is the maximum value of p such that:

$$\Pr[\neg\ (\mathrm{A} \vee \mathrm{B})] \geq \mathrm{p}$$

We recall the **union bound** for events from probability and statistics:

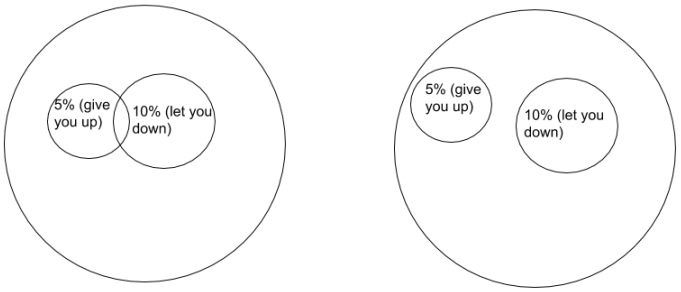
$$\Pr[\mathrm{A} \cup \mathrm{B}] \leq \Pr[\mathrm{A}] + \Pr[\mathrm{B}]$$

Therefore:

$$\begin{gathered}\Pr[\neg\ (\mathrm{A} \vee \mathrm{B})] = 1 - \Pr[\mathrm{A} \cup \mathrm{B}] \\ \Pr[\neg\ (\mathrm{A} \vee \mathrm{B})] \geq 1 - (\Pr[\mathrm{A}] + \Pr[\mathrm{B}]) \\ \Pr[\neg\ (\mathrm{A} \vee \mathrm{B})] \geq 1 - (0.05 + 0.10) \\ \Pr[\neg\ (\mathrm{A} \vee \mathrm{B})] \geq 0.85\end{gathered}$$

Hence, the probability I neither give you up nor let you down is **at least 85%**.

The diagrams below provide an illustration of this bound on the probability. If "giving you up" and "letting you down" are mutually exclusive, then it is clear based on the figure on the right that the probability of neither one occurring is 85%. Otherwise, the probability of neither one occurring will be greater than 85% (depending on how large the overlapping area is).



3 Relative Power of Representations

3.1 Review of some Knowledge Representations

3.1.1 Conjunctions

What is a conjunction?

1. A literal attribute "x_i" or its negation "$\neg x_i$"
2. An AND of literals

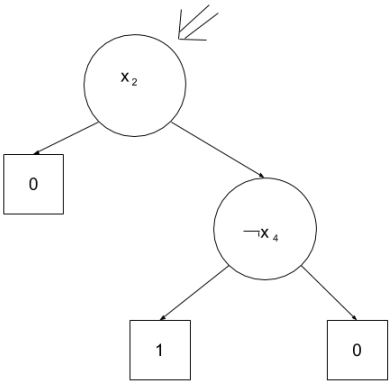
Examples: x_3, $x_1 \wedge \neg x_4 \wedge \neg x_7$

3.1.2 Decision Tree

What is a Decision Tree?

1. A tree with literal labels on internal nodes and Boolean labels on leaf nodes

Example:



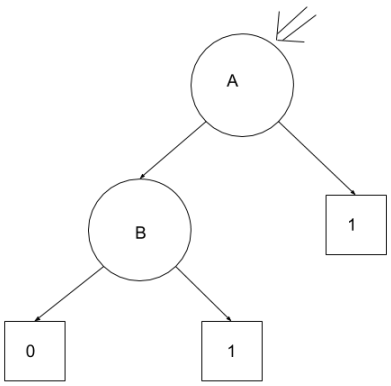
3.2 Preliminary Claims

1. There exists a decision tree representing a property that cannot be represented by a conjunction.
2. Every property that can be represented by a conjunction can also be represented by a decision tree.

Proving these two points would show that decision trees are strictly more expressive than conjunctions.

3.2.1 Example to Prove #1

The property A $\vee$ B can be expressed by the following decision tree:



Yet we cannot express this tree using just conjunctions on A/$\neg$A and B/$\neg$B. The cases below demonstrate this:

For the empty conjunction (which is considered always true by convention), there is a discrepancy when A = 0 and B = 0 since A $\vee$ B is 0 in that scenario

For the conjunction "A", there is a discrepancy when A = 0 and B = 1 because this makes A $\vee$ B = 1 while A = 0 (The conjunction "B" has the same type of problem)

For the conjunction "$\neg$A", there is a discrepancy when A = 1 and B = 1 because this makes A $\vee$ B = 1 while $\neg$A = 0 (The conjunction "$\neg$B" has the same type of problem)

For the conjunction "A $\wedge$ B", there is a discrepancy when A = 0 and B = 1 because this makes A $\vee$ B = 1 while A $\wedge$ B = 0

For the conjunction "$\neg$A $\wedge$ $\neg$B", there is a discrepancy when A = 0 and B = 1 because this makes A $\vee$ B = 1 while $\neg$A $\wedge$ $\neg$B = 0

For the conjunction "$\neg$A $\wedge$ B", there is a discrepancy when A = 1 and B = 1 because this makes A $\vee$ B = 1 while $\neg$A $\wedge$ B = 0 (The conjunction "A $\wedge$ $\neg$B" has the same type of problem)

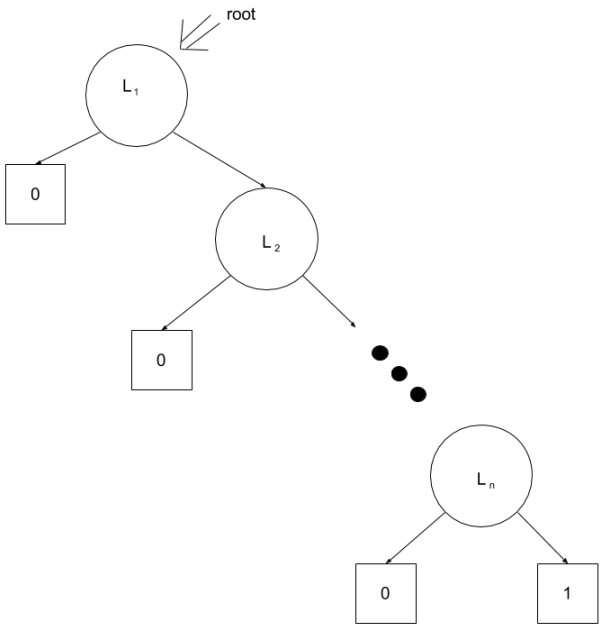
Thus, since there are no more conjunctions on A/$\neg$A and B/$\neg$B, it can be said that the property A $\vee$ B cannot be represented by a conjunction.

3.2.2 Proof of #2

We use a proof by construction:
Proof Given a conjunction of literals, $L_1 \wedge L_2 \wedge ... \wedge L_n$, we construct a decision tree in the following manner:

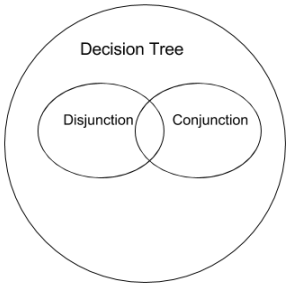
1. Root the tree at L_1
2. For each i < n, let the left child of L_i be a 0 and the right child be L_{i+1}.
3. Let the left and right children of L_n be 0 and 1, respectively.

If any literal L_i is false, then at that node of the decision tree, we'll head left and get a 0. If all literals are true, then we'll always go to the right and get a 1. Therefore, this decision tree returns true if and only if all literals are true, which is the definition of a conjunction. See the diagram below. ■



Note: swapping the 1's and 0's represents $L_1 \vee L_2 \vee ... \vee L_n$ (disjunction)

So in terms of representative power, the following diagram illustrates the relationship between conjunctions, disjunctions, and decision trees.



4 Algorithm for Learning Conjunctions

4.1 Our Goal

Given a set of n variables, x_1, x_2, ..., x_n whose assignments are drawn from a distribution $\mathcal{D}$ over X = {0, 1}^{n}, and a property c : X $\rightarrow$ {0, 1}, we seek to use a set of **labeled examples** to create a representation h : X $\rightarrow$ {0, 1} such that $\Pr_{x \in D}[c(x) = h(x)] \geq 1 - \epsilon$, i.e. we want a method of approximating c such that it guesses wrong with probability most ϵ. We restrict c such that c is always a conjunction over the n variables.

Note: While it would be nice to find c exactly, we are satisfied with a representation h described above because we do not want to have to see every data point (and the examples may not uniquely define c)

A labeled example is a tuple of the form ($\mathbf{x}^i$, b^i), where $\mathbf{x}^i$ is an **example** assignment of the variables drawn from distribution $\mathcal{D}$ and $b^i = c(\mathbf{x}^i)$ is the **label** for that example, e.g. if $c(\mathbf{x}) = x_1 \wedge \neg x_3$, the example vectors $< 0, 0, 1 >$, $< 1, 0, 0 >$, $< 0, 1, 0 >$ would have labels 0, 1 and 0, respectively, creating the tuples ($< 0, 0, 1 >$, 0), ($< 1, 0, 0 >$, 1), ($< 0, 1, 0 >$, 0).

4.2 The Elimination Algorithm

Pseudocode for the Elimination Algorithm is below:

Algorithm 1 Elimination algorithm

```
1: procedure ELIMINATION(a, b)
2:     h ← x_1 ∧ ¬x_1 ∧ x_2 ∧ ¬x_2 ∧ x_3 ∧ ¬x_3 ... ∧ x_n ∧ ¬x_n
3:     for each labeled example (x^i, b^i) do
4:         if b^i = 1 then
5:             for each literal l in h do
6:                 if l(x^i) = 0 then remove l from h
7:     return h
```

4.3 Theorem

For a fixed ϵ and δ, given a set of at least $m \geq \frac{2n}{\epsilon} \ln(\frac{2n}{\delta})$ labeled examples, the Elimination Algorithm returns a representation h of the conjunction c such that with probability $p \geq 1 - \delta$, $\Pr[c(x) = h(x)] \geq 1 - \epsilon$. Moreover, it runs in O(mn) steps.

Proof The running-time is immediately evident (The first For loop is bounded linearly by the number of labeled examples m, and the second For loop is bounded linearly by the number of attributes n, giving O(mn).

Observe that because h has all the possible literals over the n variables at the start of the algorithm, it must have all literals found in the true conjunction, c. Note that if for some literal l, $l(\mathbf{x}^i) = 0$ and $b^i = c(\mathbf{x}^i) = 1$, l cannot appear in c, since that would make c false when it is in fact true. Therefore, no literal in c can be removed by the algorithm from h, and the literals of h remain a superset of those in c at the end of the algorithm, i.e. no "good" literals are missing from h.

We must now consider the problem of "extra" literals, i.e. literals in h at the start of the algorithm which are not in c. If $l \in$ h and $l \notin c$ for some literal l, then for any variable assignment $\mathbf{x}$ where $l(\mathbf{x})$ = 0 and c($\mathbf{x}$) = 1, we'll have h($\mathbf{x}$) = 0 $\neq$ c($\mathbf{x}$).

Let us say that a literal $\tilde{l}$ is **bad** if $\Pr[\tilde{l}(\mathbf{x}) = 0 \text{ and } c(\mathbf{x}) = 1] \geq \frac{\epsilon}{2n}$. It follows that if we can generate a hypothesis h with no bad literals, then via the union bound:

$$\Pr[h(x) \neq c(x)] \leq \textstyle\sum_{\tilde{l} \in h, \tilde{l} \notin c} \Pr[\tilde{l}(\mathbf{x}) = 0,\ c(\mathbf{x}) = 1]$$
$$\Pr[h(x) \neq c(x)] \leq \#\text{ of literals in h} \times \max \Pr[\tilde{l}(\mathbf{x}) = 0 \text{ and } c(\mathbf{x}) = 1]$$
$$\Pr[h(x) \neq c(x)] \leq 2n \times \tfrac{\epsilon}{2n} = \epsilon$$
$$\Pr[h(x) = c(x)] \geq 1 - \epsilon$$

Which is our target bound. Thus, it behooves us to show that h has no bad literals with probability at least $1 - \delta$. Consider some bad literal $\tilde{l}$. If we ever draw an example for which $\tilde{l}(\mathbf{x}) = 0$ and c($\mathbf{x}$) = 1, $\tilde{l}$ will be deleted. Thus, $\tilde{l}$ is only kept after a draw if this condition is false for the drawn example.

Because we defined $\tilde{l}$ to be a bad literal, we know that $\Pr[\tilde{l}(\mathbf{x}) = 0 \text{ and } c(\mathbf{x}) = 1] \geq \frac{\epsilon}{2n}$, so $\Pr[\tilde{l}$ is kept after a draw$] = \Pr[\neg(\tilde{l}(\mathbf{x}) = 0 \text{ and } c(\mathbf{x}) = 1)] \leq 1 - \frac{\epsilon}{2n}$. Since we take m independent draws and $\tilde{l}$ must not be removed in any of them to remain in h at the end of the algorithm, we see that:

$$\Pr[\tilde{l} \text{ is kept after m draws}] \leq (1 - \tfrac{\epsilon}{2n})^m$$

Taking another union bound across all potential bad literals, we see that:

$$\Pr[\textit{Any} \text{ bad literal is kept after m draws}] \leq 2n(1 - \tfrac{\epsilon}{2n})^m$$

Since at most 2n literals (good or bad) can exist across n variables. Recall that $m \geq \frac{2n}{\epsilon} \ln(\frac{2n}{\delta})$. Plugging-in: Using the fact that $1 - x \leq e^{-x}$ for any $x \geq 0$:

$$1 - x \leq e^{-x}$$
$$1 - \tfrac{\epsilon}{2n} \leq (e^{-\frac{\epsilon}{2n}})$$
$$(1 - \tfrac{\epsilon}{2n})^m \leq (e^{-\frac{\epsilon}{2n}})^m$$
$$2n(1 - \tfrac{\epsilon}{2n})^m \leq 2n(e^{-\frac{\epsilon}{2n}})^m$$
$$\Pr[\textit{Any} \text{ bad literal is kept after m draws}] \leq 2n(e^{-\frac{\epsilon}{2n}})^m$$

Recalling that we want:

$$\Pr[\textit{Any} \text{ bad literal is kept after m draws}] \leq \delta$$

We set:

$$2n(e^{-\frac{\epsilon}{2n}})^m \leq \delta$$

And solve to get $m \geq \frac{2n}{\epsilon} \ln(\frac{2n}{\delta})$ examples, which is what the theorem states.

$$\ln(2n(e^{-\frac{\epsilon m}{2n}})) \leq \ln(\delta)$$
$$\ln(2n) + \ln(e^{-\frac{\epsilon m}{2n}}) \leq \ln(\delta)$$
$$\ln(2n) - \ln(\delta) \leq -\ln(e^{-\frac{\epsilon m}{2n}})$$
$$\ln(\tfrac{2n}{\delta}) \leq -(-\tfrac{\epsilon m}{2n})$$
$$\ln(\tfrac{2n}{\delta}) \leq \tfrac{\epsilon m}{2n}$$
$$\tfrac{2n}{\epsilon}\ln(\tfrac{2n}{\delta}) \leq m$$
$$m \geq \tfrac{2n}{\epsilon}\ln(\tfrac{2n}{\delta})$$

■

5 PAC-Learning Model (in-General)

5.1 Definition

Let $C = \cup_{n \in \mathcal{N}} C_n$ be a class of representations such that C_n is a class of representations on n attributes.

We say that C is **PAC-Learnable** if there is an algorithm that (for a given $0 < \delta < 1/2$, $0 < \epsilon < 1/2$) if given access to a set of labeled examples of a property $\mathrm{c} \in \mathrm{C}_n$ for some n (and the examples are drawn independently from a fixed, arbitrary distribution D), execute in time polynomial in n, $\frac{1}{\delta}$, $\frac{1}{\epsilon}$ and $|c|$ and return with probability at least $1 - \delta$ returns a representation h of c which is approximately correct, i.e. $\Pr[\mathrm{h(x)} = \mathrm{c(x)}] \geq 1 - \epsilon$.