CSE 513T Theory of Artificial Intelligence & Machine Learning Sep 6th, 2016

Lecture 3

Lecturer: Brendan Juba *Scribe: Xuezhu Li, Editor: James DiLorenzo*

1 Today

1. Complexity of knowledge representations
2. Efficient computation algorithms and computational problems

2 Reminder: PAC-learnable

Let $C = \cup_{n \in N} C_n$ be a class of representations where C_n consists of representations on n attributes. We say that C_n is **PAC-learnable** if there exists an algorithm that, given as input parameters $\varepsilon \in (0, 1/2)$ and $\delta \in (0, 1/2)$, and given access to labeled examples drawn independently and identically from an arbitrary distribution D over settings of the n attributes, runs in time polynomial in n, $1/\varepsilon$, $1/\delta$, and the size of representation $c \in C$ used to label the examples, and with probability $\geq 1 - \delta$ over the data returns a representation h such that $Pr_{x \in D}[h(x) = c(x)] \geq 1 - \varepsilon$.

3 Warm-up Question

3.1 Description

Last time, we saw an algorithm for PAC-learning *conjunctions* — ANDs of "literals" (attributes or their negations).We also discussed *disjunctions* — ORs of literals.

Show how, using the algorithm for PAC-learning conjunctions, you can obtain an algorithm for PAC-learning disjunctions.

3.2 Idea

- Reverse (negate) labels in data.
- Take resulting conjunction:
  - switch ANDs to ORs
  - negate the literals
- Use the Elimination Algorithm for PAC-Learning conjunctions on the resulting conjunction (See Lecture 2)

Rationale: This is an application of de Morgan's Laws, which state that $\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q)$

1. Run time: extra $O(n)$ steps per example — still polynomial in n.
   We know by de Morgan's law that if originally the data was labeled by a disjunction

   $$l_1 \vee l_2 \vee ... \vee l_s \ (size \approx s)$$

   Then by negating the labels, we obtain data labeled by

   $$\neg(l_1 \vee l_2 \vee ... \vee l_s) = \neg l_1 \wedge \neg l_2 \wedge ... \wedge \neg l_s$$

   which is a conjunction of size s. So the algorithm still takes time bounded by a polynomial in the size of the original disjunction.

2. The resulting data is labeled by a conjunctions, so with probability $\geq 1-\delta$ over the data, we get a conjunction h such that
$$Pr_{x\in D}[h(x) = \neg c(x)] \geq 1-\varepsilon$$
therefore,
$$Pr_{x\in D}[\neg h(x) = c(x)] \geq 1-\varepsilon$$

This algorithm is just applying de Morgan's law to obtain a disjunction from $\neg h(x)$.
This illustrates "reductions", converting one problem into another by some algorithm conversion.

4 DNF (Disjunctive Normal Form)

4.1 Description

DNF An OR of ANDs of literals.

Examples

$(x_1 \wedge \neg x_3 \wedge x_4 \wedge x_{11}) \vee (x_2 \wedge x_3 \wedge \neg x_{42}) \vee (\neg x_4) \vee ...$

$(x_1 \wedge \neg x_3 \wedge x_4 \wedge x_{11})$ — "term" of size 4

$(x_2 \wedge x_3 \wedge \neg x_{42})$ — "term" of size 3

4.2 DNF Expression

4.2.1 DNFs can also capture both conjunctions and disjunctions

Disjuncitons many terms of size 1

Conjunctions one term containing the desired conjunction

4.2.2 Is there any property that cannot be expressed by a DNF?

Proposal No. List out all elements in the space as a DNF.

Concretely write the truth table.

For each line where the property is true, write a conjunction that picks out only that line.

OR them together.

Example: Implication $x \implies y$

Truth Table:

P	Q	P $\implies$ Q
0	0	1
0	1	1
1	0	0
1	1	1

can be represented in DNF as

$$(\neg x \wedge \neg y) \vee (\neg x \wedge y) \vee (x \wedge y)$$

4.3 Size of representations(in bits, etc)

4.3.1 How can I represent a decision tree by a "small"(related size) DNF?

Proposal Start at leaves, take leaves that are true, follow paths back to root. Take the conjunction of labels on that path and add them to the current DNF.

(Note that do not add negated false conjunctions to DNF.)

DNF contains a disjunction that picks out every branch that leads to 1. So whenever the tree would reach a true branch, the DNF evaluates to true.

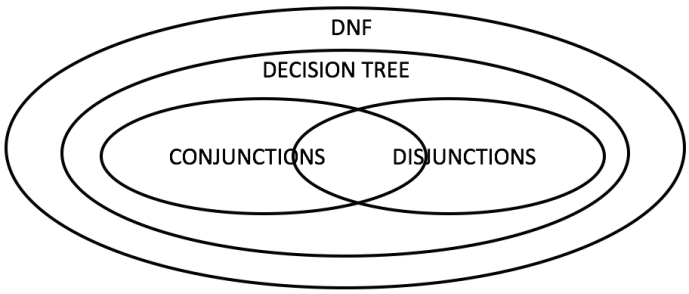
Conversely, if the DNF is true, one of these conjunctions needs to be true. Therefore, the decision tree would also label it to be true.

So, the tree and the DNF have identical truth values.

We have a conjunction of size equal to the length of the path for each true leaf of the tree.

$$\begin{aligned} size(in\ literals) &\leq \sum_{leaves} (length_of_path) \\ &\leq (number_of_nodes)(number_of_nodes) \\ &\leq (number_of_nodes)^2 \text{ – At worst a polynomial increase} \end{aligned}$$

So can represent decision tree efficiently (with $\leq$ polynomial increase in size) as a DNF.



It turns out that the reverse is not true. For example, the following DNF of size s (number of literals) requires a decision tree of size $2^{\Omega(s)}$:

$$Tribes_{2,s/2} = (x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee ... \vee (x_{s-1} \wedge x_s)$$

(See Homework 1 for proof)

In general, we find that any two "intuitively reasonable" representations:

1. differ only in size to a polynomial amount, but
2. may differ by arbitrary polynomial amount.

Also motivated by our tolerance to arbitrary polynomial dependence on the representation size in our algorithms' running time.

5 Efficient Algorithms

We take the definition of an **"efficient algorithm"** to mean one that runs in time polynomial in its input size.

Why? — 3 reasons:

1. Mathematical convenience

   Polynomial time is the smallest class of functions that

   (a) contains all linear-time functions;

   (b) is closed under use of subroutines (composition).

2. Experience in practice

   Polynomial time algorithms have to exploit some kind of nontrivial "structure". We find that when this structure/insight is fully exploited, then we generally end up with algorithms that are efficient in practice.

3. Invariance

   Every currently *viable** model of computation can be simulated by a RAM machine (or Turing Machine, or ...) with only a polynomial slow-down in running time (But, in general with an arbitrary polynomial slow down).

   Thus, even as new architectures are introduced (e.g. multi-core GPU, etc), our standard for what constitutes a "good algorithm" remains fixed.

   $\star$ Quantum computing may provide super-polynomial speedup, but large scale generalized Quantum Computing doesn't exist (yet).

6 Computational Problems

Definition 1 (Relation Problem) *For sets X, Y ("inputs" and "outputs"), and a relation $R \subseteq X \times Y$, the corresponding relation problem is "given $x \in X$, find $y \in Y$ such that $(x, y) \in R$".*

Definition 2 (Consistency Problem) *Let $X = \cup_{n \in N} X_n$, where X_n consists of lists of labeled examples on n attributes. We will say that a list $\bar{x} \in X_n$ is **consistent** with a representation c if for every labeled example $(x, b) \in \bar{x}, c(x) = b$.*
If Y is a set of encodings of a class of representations C, then the relation problem

$$Consis_C = \{(\bar{x}, y) \in X \times Y : y \text{ represents } c \in C \text{ that is consistent with } \bar{x}\}$$

Note that the elimination algorithm is a poly-time algorithm for the problem $Consis_{Conjunctions}$.