CSE 513T Theory of Artificial Intelligence & Machine Learning January 23, 2018

Lecture 3

Lecturer: Brendan Juba *Scribe: Stephen Mell*

Contents

1 Warm-Up **1**

2 Relative Power of Representations **1**

3 Relation Problems **2**

4 NP-Search Problems and NP-Completeness **3**

1 Warm-Up

Problem

Last lecture, we saw the Elimination Algorithm for PAC-learning conjunctions (ANDs of literals). Show how to use the Elimination Algorithm as a subroutine to learn disjunctions (ORs of literals).

Solution

Idea Negate the attributes in the examples and the labels, then return the disjunction on the same set of literals.

De Morgan's Laws

$$\neg(l_1 \wedge l_2 \wedge \ldots \wedge l_k) \equiv \neg l_1 \vee \neg l_2 \vee \ldots \vee \neg l_k$$

$$\neg(l_1 \vee l_2 \vee \ldots \vee l_k) \equiv \neg l_1 \wedge \neg l_2 \wedge \ldots \wedge \neg l_k$$

Steps

1. Since we suppose the target C is given by a disjunction $(l_1 \vee l_2 \vee \ldots \vee l_k)$ when we negate all of the attributes and the label, our labels are given by $\neg(\neg l_1 \vee \neg l_2 \vee \ldots \vee \neg l_k) \equiv l_1 \wedge l_2 \wedge \ldots \wedge l_k$. The Elimination Algorithm finds a conjunction $l'_1 \wedge l'_2 \wedge \ldots \wedge l'_{k'}$.
2. The disjunction $l'_1 \vee l'_2 \vee \ldots \vee l'_{k'}$ is correct on $(\mathbf{x}, b)$ if and only if $l'_1 \wedge l'_2 \wedge \ldots \wedge l'_{k'}$ is correct on $(\neg\mathbf{x}, \neg b)$. Since $l'_1 \wedge l'_2 \wedge \ldots \wedge l'_{k'}$ has an error rate of less than ϵ on $(\neg\mathbf{x}, \neg b)$, $l'_1 \wedge l'_2 \wedge \ldots \wedge l'_{k'}$ has an error rate of less than ϵ on $(\mathbf{x}, b)$.

This is an example of a *reduction*: transforming learning disjunctions into learning conjunctions.

2 Relative Power of Representations

Recall We were relating the size of decision tree representations and DNF representations. (*Disjunctive normal forms* are ORs of ANDs of literals.)

Claim 1 There is a DNF of size s that requires a decision tree of size $2^{\omega(s)}$ to represent. For even s, $(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \ldots \vee (x_{s-1} \wedge x_s)$ is an example of a $Tribes_{2,s/2}$ DNF. For proof, see homework 2.

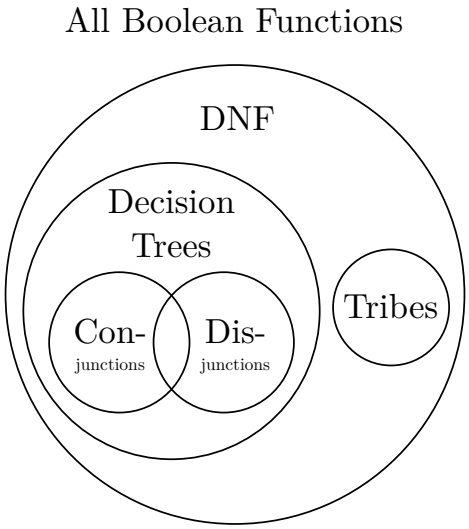

Figure 1: We tolerate polynomial size increases, since, for our purposes, "efficient algorithms" only need to have some polynomial dependence on the underlying parameters.

Claim 2 Every decision tree of size s can be represented by a DNF of size $O(s^2)$. Consider each of the leaves which take value "true". Take an AND of the literals describing which branch of the tree is taken to reach that the true leaf, and the resulting conjunction is a term. Taking the AND of all of these terms gives us the desired DNF. Each term has no more than s literals, and there are no more than s leaves, so the overall DNF has size no more than $(leaf\ count) \cdot (conjunction\ size) = s^2$.

Example Suppose we have some algorithm that runs in polynomial-time in the size of the DNF and we are interested in working with a decision tree. If we convert the tree to a DNF, we still obtain an algorithm that is polynomial-time in the size of the decision tree.

Reasons for Focusing on Polynomial Time

1. Convenience — polynomials compose, so we can assume polynomial-time subroutines.
2. Experience — polynomial-time algorithms need an idea which can usually be exploited to give more refined algorithms that are efficient in practice.
3. Model-invariance — every currently viable model of computation (we exclude quantum computation) can be simulated in polynomial-time on a RAM machine, up to polynomial-factor slowdowns.

3 Relation Problems

For sets X and Y of inputs and outputs, respectively, and a relation $R \subseteq X \times Y$, the corresponding worst-case problem is, "given some $x \in X$, find $y \in Y$ such that $(x, y) \in R$".

Let $X = \{X_n\}_{n=1}^{\infty}$ where each X_n consists of lists of labeled examples on n attributes. Let Y be the encodings of conjunctions on arbitrary numbers of attributes. The Elimination Algorithm is solving the relation problem $Consis_{Conj} = \{(\mathbf{x}, c) \in X \times Y : \mathbf{x}$ is *consistent* with $c\}$. We say the list of labeled examples $\mathbf{x} = [(\mathbf{x}^{(1)}, b^{(1)})\ (\mathbf{x}^{(2)}, b^{(2)}) \ldots (\mathbf{x}^{(m)}, b^{(m)})]$, where each $\mathbf{x}^{(j)}$ is an n-attribute vector for some n, is *consistent* with the conjunction c if for every j-th example in $\mathbf{x}$, $c(\mathbf{x}^{(j)}) = b^{(j)}$.

Of course, PAC-Learning is an average-case task (in that it refers to a probability distribution) and this is a worst-case problem, so this is *not* the same as the PAC-Learning task.

Showing certain representations aren't learnable is equivalent to showing that a learning algorithm does not exist, so we need to be able to show that algorithms don't exist. Reductions give a means to achieve this. If a hypothetical learning algorithm lets us solve a problem that we think is too hard, that implies that the learning algorithm should not be possible. The problems we don't think we can solve are *NP-Complete* problems.

4 NP-Search Problems and NP-Completeness

Definition 1 *A relation problem $R \subseteq X \times Y$ is an* NP-search problem *if*

1. *R is* polynomially balanced *— there exists a polynomial $p(n)$ such that for every x such that $(x, y) \in R$ for some y, there is some y' such that $|y'| \leq p(|x|)$ (sizes in bits) and $(x, y') \in R$.*
2. *R is* efficiently verifiable *— there is a polynomial-time algorithm for computing the boolean function*

$$R(x, y) = \begin{cases} 1 & \text{if } (c, y) \in R \\ 0 & \text{otherwise} \end{cases}$$

Example If $\mathscr{C}$ is a class of efficiently evaluable representations, then the problem

$$Consis_{\mathscr{C}}^{p(n)} = \{(\mathbf{x}, c) : c \in \mathscr{C} \text{ is of size no more than } p(n) \text{ and is consistent with } \mathbf{x}\}$$

is an NP-search problem because:

1. If $(\mathbf{x}, c) \in Consis_{\mathscr{C}}^{p(n)}$, then we're promised $|c| \leq p(n) \leq p(|\mathbf{x}|)$, so p suffices for this polynomial bound.
2. We can choose the algorithm:

   for $\mathbf{x}^{(j)}$ in $\mathbf{x}$ **do**
   if $c(\mathbf{x}^{(j)}) \neq b^{(j)}$ **then**
   return 0
   end if
   end for
   return 1

   Since $\mathscr{C}$ is efficiently evaluable, the check runs in polynomial-time, and since we need to conduct a polynomial number of checks (no more than $|\mathbf{x}|$), the overall running time is polynomial as a function of $|c|$ and $|\mathbf{x}|$.

Definition 2 *An NP-search problem R is* NP-complete *if for every other NP-search problem S, there is an algorithm that solves S in polynomial-time using a polynomial-time algorithm for R as a subroutine.*

Note Since there are many NP-search problems that resist the design of efficient algorithms, we don't believe there are polynomial-time algorithms for NP-complete problems.

Definition 3 *A* boolean circuit *is given by a directed, acyclic graph in which:*

1. *The sources are labeled either by attributes $x_1, x_2 \ldots x_n$ or constants* 1 *or* 0*.*

2. *The k sinks are labeled $y_1, y_2 \dots y_k$.*

3. *The internal nodes are labeled by $\wedge, \vee$, or $\neg$, where $\wedge$ and $\vee$ nodes have in-degree 2 and $\neg$ nodes have in-degree 1.*

The circuit computes a function $f : \{0,1\}^n \to \{0,1\}^k$ in the natural way.