Homework 1

Due: February 13, 2018 at 7:00PM

Written Questions

Problem 1: Decision Trees

Trapped in the Upside Down, Will Byers could only communicate with his mother Joyce in the real world by activating a series of lights strung up in his house. Each light was assigned a letter of the alphabet, and Will communicated words by activating lights corresponding to each letter in series. After a near death experience with the Demogorgon, Joyce decided the current communication paradigm was too inefficient. If all lights can be used simultaneously, there are many more available permutations of lights for communication. Joyce has taken down the lights labelled A through Z, and has raised lights labelled numerically 1 to d. Each light can exist in one of two states: on or off. Additionally, Joyce deduced that Will only ever communicates two distinct messages: RUN and STAY. In other words, Joyce must deal with a binary classification problem to construct a decision tree.

Part 1

As Will is still in the Upside Down, he cannot communicate with Joyce directly to verify the meaning of his code. Joyce hopes to employ PAC learning to understand Will's code. For every code Will sends, Joyce observes whether the code means STAY or RUN.

1. Suppose a decision tree is constructed by branching on the attributes in sequential order (1 to d). How many possible decision trees are there of full depth? In other words, how many coding schemes could Will create using all d lights? (We're looking for a simple upper bound, not an exact count of all distinct functions.)

   Solution:

   We are told that the decision tree is constructed by branching on attributes in sequential order, so there is only one structure for the decision tree. Each branch results in two new nodes, so each subsequent layer in the tree has twice as many nodes as the previous layer. At depth d, the final layer will have 2^d leaves. Each leaf can be classified as STAY or RUN, which gives us two options for each leaf. This leads to a total of 2^{2^d} possible decision trees.

2. Expressed in terms of d, δ and ϵ, how many observations are needed to conclude a hypothesis is probably, approximately correct? Is PAC learning feasible (polynomial) for this problem?

   Solution:

   We are considering the realizable case. The PAC learning bound for the realizable case is:

   $$\frac{log(\frac{|H|}{\delta})}{\epsilon}$$

   Plugging in the size of our hypothesis space:

   $$\frac{log(\frac{2^{2^d}}{\delta})}{\epsilon} = 2^d \frac{-log(\delta)}{\epsilon}$$

   The PAC learning bound is $O(2^d)$ which is exponential in terms of d. PAC learning is not feasible for this problem.

Part 2

Will is considering some alternative coding schemes, and wants to know how they would translate into decision trees. For each of the following coding schemes, how many leaves would there be in the smallest decision tree? Express in terms of d.

1.
$$\begin{cases} \text{At least one light is on} \implies \text{RUN} \\ \text{No lights are on} \implies \text{STAY} \end{cases}$$

Solution:

Branch on d lights in sequential order. If the light is on, we know that the answer is RUN, so this would be a leaf. If the light is off, we must branch to the next light. Therefore, every light will have a leaf and a non-leaf child except for the last light, which will have two leaves.

2.
$$\begin{cases} \text{An odd number of lights are on} \implies \text{RUN} \\ \text{An even number of lights are on} \implies \text{STAY} \end{cases}$$

Solution:

Given some sequence of lights, the state of the next light will always be determinate in whether an even or odd amount of lights are currently lit. In every scenario, the state of ever light must be known in order to determine if the number of lit lights is even or odd. Therefore, branching will always occur, and we will obtain a tree of full depth. The tree of full depth will have 2^d leaves (doubling every unit of depth).

Problem 2: Partial Orderings

Background Information: A *partial ordering* (denoted $\preceq$) is a binary relation over a set S that satisfies the following properties:

- *Reflexivity*: $\forall s \in S, \quad s \preceq s$
- *Antisymmetry*: $\forall s, t \in S, \quad s \preceq t, t \preceq s \implies s = t$
- *Transitivity*: $\forall r, s, t \in S, \quad r \preceq s, s \preceq t \implies r \preceq t$

A *total ordering* differs from a partial ordering in providing an additional guarantee that all elements within S can be compared in the relation. As an example, the $\leq$ relation is a total ordering on the real numbers. Any two numbers a, b may be compared such that either $a \leq b$ or $b \leq a$. Additionally, the $\leq$ relation satisfies the conditions of reflexivity, antisymmetry and transitivity stated above. *Note: All total orderings are also partial orderings.*

One example of a partial ordering is the $\subseteq$ relation. The properties of reflexivity, transitivity and antisymmetry are inherent to the $\subseteq$ relation. However, given two sets A and B, there is no guarantee that either $A \subseteq B$ or $B \subseteq$ A. Now, consider a classification problem defined as follows:

Input Space: The set of all possible n-element permutations.
Hypothesis Space: Is the given ordering consistent with a hidden partial ordering?

The input space consist of all the possible ways that a given set of n elements can be arranged. The hypothesis space maps each permutation of the elements to 0 or 1, depending on whether or not the permutation is consistent with the partial ordering.

1. What is the hypothesis space H of representations the learner is searching?

   Solution:

   One way to think about this problem is thinking about it from a machine learning perspective. Our training data (or rather all of our data) is all of the n-element permutations. For each permutation, we are also given a label that indicates if this permutation is consistent with the hidden partial ordering.

   Our goal is to learn this partial ordering from the permutations and corresponding labels. Therefore, the hypothesis space H of representations is merely the set of all possible partial orderings.

2. Show that $|H|$ is between $n!$ and $3^{(n^2)}$. *Hint: Each $h \in H$ may be represented as a Directed Acyclic Graph (DAG).*

   Solution:

   On the lower bound, we know that any total ordering will be a partial ordering as well. As a total ordering will order all elements within the set, it is essentially a permutation. There are $n!$ total orderings, so there are at least $n!$ partial orderings.

   For the upper bound, think about each n element as a node in a DAG. There are (on the order of) n^2 pairs of nodes. (The exact number would be $\frac{n(n-1)}{2}$) Reflexivity may be handed separately by drawing an edge from every node to itself. Between any pair of elements i and j, you have three possibilities:

   - A directed edge from i to j.
   - A directed edge from j to i.
   - No edge between i and j.

Intuitively this makes sense because in a partial ordering, you can either have a relationship between two elements or no relationship between two elements (eg. if the partial ordering is subsets, comparing $\{a, b\}$ with $\{c, d\}$ would give us no relationship between two elements). Note that this collection of 3 states also enforces antisymmetry. This gives us a total of $3^{(n^2)}$ possible partial orderings. Note that this bound is not tight because it allows for cycles and other violations of transitivity; however, every partial ordering is indeed within this set.

3. Show that the amount of data needed to ensure the ERM solution is PAC is polynomial in n. **Solution:** The dependence on the hypothesis space is $log(|H|)$, which has an upper bound of $log(3^{(n^2)}) = O(n^2)$, so this is polynomial in n.

4. Describe an algorithm you could use to efficiently find an ERM solution for a given sample of observations.

   Solution:

   We begin by looking at the positive instances in the training set. For each pair of nodes i, j, we check if we always have i before j or j before i in all of the positive examples. This allows us to build a corresponding partial order. This collection of constraints will always adhere to the definition of a partial order:

   - If for all positive instances, $i \preceq j, j \preceq k$, it follows that $i \preceq k$ for all positive instances.
   - It cannot be simultaneously true that $i \preceq j$ and $j \preceq i$ for any given instance.

   We are given the true hypothesis is within our hypothesis space, so the ERM should correctly classify all data in our sample. Our partial ordering is correct in positive classification by construction. We will now proof our partial ordering correctly classifies all negative instances.

   Proof. Suppose our ERM partial ordering incorrectly classifies a negative instance as positive. The negative instance must therefore satisfy all constraints that are shared among all of the positive instances. This indicates that there are no constraints that uniquely differentiate the set of positive instances from this negative instance. As we know there exists a hidden partial ordering that correctly classifies our sample, this is a contradiction. Therefore, our ERM partial ordering cannot incorrectly classify a negative instance. □

   Constructing the ERM partial ordering is polynomial over the sample size $|S|$ and the size of the permutatios n. Determining whether to include each potential i, j pairing in the partial ordering occurs in $|S|$: checking whether a particular instance satisfies the rule occurs in constant time, and there are at most $|S|$ positive instances to check. There are $O(n^2)$ rules to check, so the entire algorithm runs in $O(|S|n^2)$.