HW 4 - Learning Theory

Michael Young - u0742759

1. Pac Learnability of Depth Limited Decision Trees

1.a) What is $S_n(0)$? That is, how many trees of depth 0 exist?

With a depth of 0, we can have 2 unique trees, because we can either label all examples as 1 (True) or 0 (False).

1.b) What is Sn(1)? That is, with n features, how many trees of depth 1 exist?

With a depth of 1, it means we are splitting on 1 feature. There are n such features we could split on. After this split, we have 2 branches, whose leaves we can label as either [1,1], [1,0], [0,1], or [0,0], for a total of 4 ways. Thus, $S_n(1) = 4n$.

Note here that I am not considering "mirror image" trees as additional unique trees. I.e. if the branches have values [1,0] and the labels respectively are [1,0], then I consider this to be the same tree if the branches were [0,1] and the corresponding labels were [0,1]. If instead I include these trees as additionally unique, that would just be another factor 2 in my counting, which wouldn't change the overall big O complexities that we're concerned with.

1.c) Suppose you know the number of trees with depth i, for some i. This quantity would be Sn(i) using our notation. Write down a recursive definition for Sn(i + 1) in terms of n and Sn(i). For this expression, you can assume that we are allowed to the use same feature any number of times when the tree is constructed.

To explore this, let's calculate the # of trees for a few example depths:

$$S_n(0) = n^0 2^1$$
$$S_n(1) = n^1 2^2$$
$$S_n(2) = n^3 2^4$$
$$S_n(3) = n^7 2^8$$

Here, the 2 term takes care of the combinations of leaves for each depth, and the n term takes care of the nodes of the trees at each depth, which can always take any number of n values. By examining this, we can clearly identify the pattern:

$$S_n(i+1) = S_n(i)^2 n$$

Note that the base case for this recurrence is $S_n(0) = 2$.

1.d) Recall that the quantity of interest for PAC bounds is the log of the size of the hypothesis class. Using your answer for the previous questions, find a closed form expression representing logSn(k) in terms of n and k. Since we are not looking for an exact expression, but just an order of magnitude, so you can write your answer in the big O notation.

By either solving the recurrence relation identified above (by plugging into Wolfram alpha because I am lazy) or examining more closely the pattern from 1.c, we get the closed form expression:

$$S_n(k) = n^{2^k - 1} 2^{2^k}$$

We then take the log of this expression to get the term we are interested in for PAC learnability (If I remember correctly, the log associated with PAC learnability is the natural log, so that is what I use below):

$$log(n^{2^k - 1} 2^{2^k})$$

Simplifying down to big O notation, we get:

$$\begin{aligned} &= (2^k - 1)(log(n))(2^k)(log2) \\ &\approx O(2^k log(n)) \end{aligned}$$

2.a) With finite hypothesis classes, we saw two Occam's razor results. The first one was for the case of a consistent learner and the second one was for the agnostic setting. For the situation where we are given a dataset and asked to use depth-k decision trees as our hypothesis class, which of these two settings is more appropriate? Why?

The consistent learnability setting applies when we have a hypothesis that will be 100% consistent with the training set, meaning it can learn the training set perfectly. The agnostic setting is used when we don't have this guarantee of consistency. In the case where we are using a depth-k decision tree, the agnostic setting is more appropriate, because with a depth limited tree you're not guaranteed to learn the training set perfectly, like you are when you are allowed a full tree (assuming no noisy data).

2.b) Using your answers from questions so far, write the sample complexity bound for the number of examples m needed to guarantee that with probability more than 1 − δ, we will find a depth-k decision tree whose generalization error is no more than ε away from its training error.

Using the agnostic setting equation we achieved using Occam's razor, and plugging in our answer from 1.d we get:

$$m \geq \frac{1}{2\epsilon^2}(2^k log(n) + log(\frac{1}{\delta}))$$

3. Is the set of depth-k decision trees PAC learnable? Is it efficiently PAC learnable?

For a set of hypotheses to be PAC learnable, they need to be polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}, n, size(H)$. Referring to the result obtained in 2.b, we see that yes, the set of depth-k decision trees is PAC learnable. However, I wavered on this answer because of the 2^k term which is not polynomial in k. This seems to violate the polynomial requirement for PAC learnability, doesn't it? I've decided that it doesn't, because for any specific set of depth-k decision trees, the k is fixed, but the # of features n can still vary. We are concerned with the polynomiality of n, not k.

For this set to also be efficiently PAC learnable, there needs to be an algorithm L that can produce the hypothesis in time polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}, n, size(H)$. I argue that the ID3 algorithm has a time complexity that satisfies this requirement. Where m are training examples and n is the # of attributes or features, the ID3 algorithm runs in $O(mn^2)$, which is polynomial in n.

4. Suppose the number of features we have is large and the depth limit we are considering is also large (say, in the thousands or more). Will the number of examples we need be small or large? Discuss the implications of the sample complexity bound from above.

Referring back to $m \geq \frac{1}{2\epsilon^2}(2^k log(n) + log(\frac{1}{\delta}))$, we see that k is present in an exponent (2^k) while n is present in a log. Therefore, as k grows, m increases much faster than if n grows. So in this instance where k is large but n is also large, we'll definitely need a lot of examples. However, if k is relatively small but n is large, because n is in a log, we should still be able to get away with relatively few samples.

2. Shattering

Does the set of templates Hn shatter the set Xn? Prove your answer.

To gain an intuition for this problem, let's start by considering a concrete example, where $n = 2$.

$$X_n = \{01, 10, 00, 11\}$$
$$H_n = \{01, 10, 00, 11, -0, -1, 0-, 1-, --\}$$

Generally, in order for X_n to be shattered by H_n, we would need to show that there is no labeling of the set X_n that cannot be correctly classified by the members of H_n. There are 2 cases that I can immediately identify where we show that no members of H_2 can correctly fit X_n:

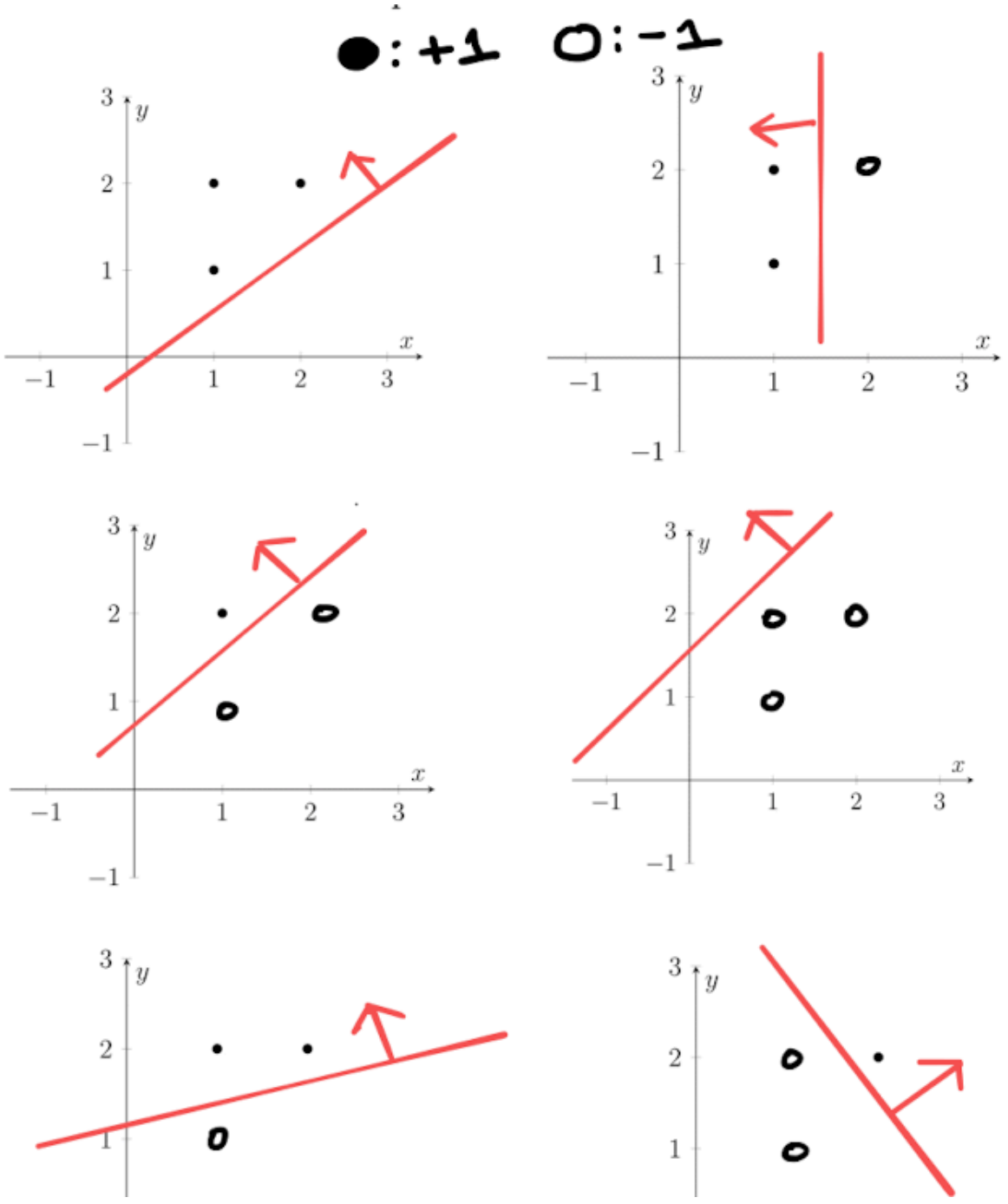
$$(1) X_n = \{01 : -1, 10 : -1, 00 : +1, 11 : +1\}$$
$$(2) X_n = \{01 : -1, 10 : -1, 00 : -1, 11 : -1\}$$

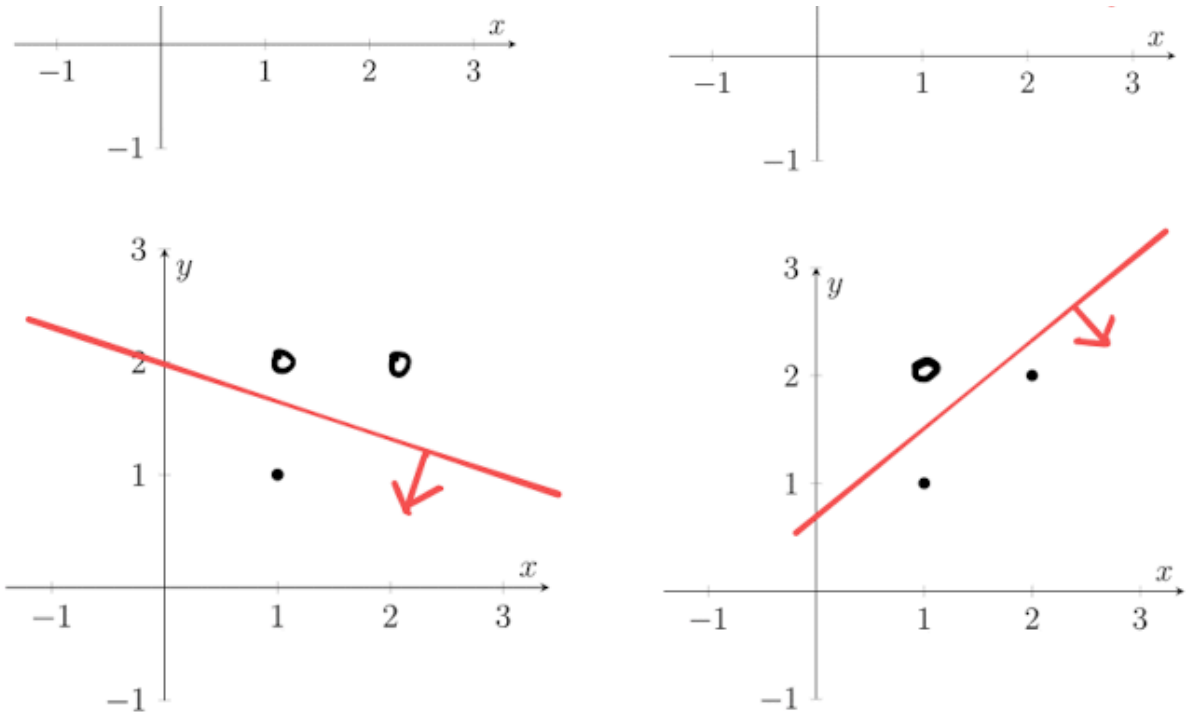
For case (1), in order to match 00 and 11 we could use $--$, but because $-$ always matches, it would incorrectly identify 01 and 10 as $+1$. For case (2), we see that by labeling all members -1, there is no template in H_2 that can handle this. All members of H_2 match at least one member of X_2, thus they could never correctly classify all negative labels.

These observations generalize to cases greater than $n = 2$. For case (1), if we label the all $1's$ or all $0's$ members of X_n as $+1$, we would need to use the all $-'s$ member of H_n to correctly classify, which means it would fail on any other -1 labeled members of X_n. For case (2), by definition, there will be no member of H_n that can correctly classify all members of X_n as -1 because every member of H_n will match with at least 1 member of X_n.

3. VC Dimension

1. Assume that the three points below can be labeled in any way. Show with pictures how they can be shattered by a linear classifier. Use filled dots to represent positive classes and unfilled dots to represent negative classes.





2.a Show that the VC dimension of H^2_{rec} is 4.

To show that the VC dimension of H^2_{rec} is 4, we must show that there is some set of 4 points that is shattered by axis aligned rectangles, then show that there are no set of 5 points that are.

When considering 4 points, let's identify some cases that aren't shattered.

(1) If 3 of the points are collinear, then no matter where the 4th point is, the set is not shattered, because if the 2 outside points are positive and the inside point is negative, there's no way for a rectangle to exclude the inner point.

(2) Similarly, if 3 points make a triangle, and there's a point inside this triangle, this is also not shatterable. If all points of the triangle are positive, and the inner point is negative, there's no way for the rectangle to only envelope the outer points and exclude the inner. Both of these examples show that sets are not shatterable by this hypothesis space when the rectangles cannot cover every possible subset of points.

To show that there is some set of 4 points that is shattered by the infinite set of axis aligned rectangles, consider a diamond shape - equivalent to points located at [0,1],[1,0],[-1,0], and [0,-1] in a standard cartesian coordinate plot. This set is shattered by axis aligned rectangles, because the rectangles can cover every subset of the 4 points (i.e. $\{[0,1],[0,-1]\}, \{[0,1],[0,-1],[-1,0]\}$ etc...).

To show that 5 points are not shattered by the set of axis aligned rectangles, let's imagine if we were to label the top-most point, the bottom-most point, the left-most point, and the right-most point as $+$ examples. We would necessarily have to draw a rectangle around these points to label them correctly. By drawing a rectangle around these 4 (sometimes 2 or 3) points, we essentially draw a bounding box around the entire set of points, thus if any of the other points are labeled negative, we

would incorrectly classify them, because they would lie inside of the rectangle. This logic holds for any arrangement of 5 points, thus the set of 5 points is not shattered by axis aligned rectangles.

Therefore, the VC dimension of H_{rec}^2 is 4.

2.b Generalize your argument from the previous proof to show that for d dimensions, the VC dimension of H_{rec}^d is 2d.

To show this, let's consider again the argument where we labeled the top-most, bottom-most, right-most, and left-most points as positive. For 2 dimensions, this essentially boiled down to finding 4 points, the max/min of x_1 and the max/min of x_2. Any point added outside of these 4 points proved to be unshatterable. Let's consider H_{rec}^3. For 3 dimensions, the same "bounding box" argument applies, only now we are considering 6 points: the max/min of x_1, x_2, and x_3. Any additional point will necessarily lie within this cubic bounding box, thus making 6 + 1 unshatterable. It follows that we can construct such a hyper bounding box for any number of d dimensions consisting of $2d$ points, where the $2d$ +1 set would then be unshatterable. Thus, the VC dimension of H_{rec}^d is $2d$.

3.a Show that for a finite hypothesis class C, its VC dimension can be at most log2 (|C|). (Hint: You can use contradiction for this proof. But not necessarily!)

Ok, let's consider a set of 3 examples. Each of these examples can be labeled in 2 different ways, meaning this set of 3 examples has $2^3 = 8$ distinct labelings. We have a finite hypothesis class C consisting of $|C|$ functions. Let's say $|C| = 4$. These functions could yield at most 4 different labelings. If we return to our example of 3 data points, we saw that there were 8 distinct labelings, which our 4 functions can't cover. It's clear that for 3 examples, we'll need at least 8 functions in our hypothesis class to cover all of the potential labelings.

Taking this more generally, we see that when n is our number of points, that there are 2^n potential labelings of these points, and thus 2^n functions needed to cover these distinct labelings. Here $|C| = 2^n$ and $log_2(|C|) = n$, which is the number of points it can shatter i.e. it's VC dimension. If there are more labelings of points than functions, then there may be a labeling that is not covered by a function. Thus, we see that the VC dimension of a finite hypothesis class C, can be at most $log_2(|C|)$. We say "can be at most" b/c it's perfectly possible that we have a "bad" hypothesis class C with many functions, but because they are "bad" functions they can't shatter the max of $log_2(|C|)$ points.

3.b Find an example of a class C of functions over the real interval X = [0, 1] such that C is an infinite set, while its VC dimension is exactly one.

The infinite set of functions C that has a VC dimension of exactly one would be the set of points in the interval $[0, 1]$ where we classify a point as $+$ if it matches our point in C and $-$ otherwise. As an example, say we have a point at 0.5 labeled "+". A function in our class C that would correctly classify this would be the matching point 0.5. Alternatively, if we labeled the point at 0.5 "-", our function from C could be any other point in that interval and correctly classify the point at 0.5, because it would label any point it didn't match as "-".

If we add another point, then our infinite set of functions will fail to shatter, because 1 point cannot simultaneously be equal to 2 or more points.

3.c Give an example of a finite class C of functions over the same domain X = [0, 1] whose VC dimension is exactly log2(|C|).

Say we have n points over the domain X. These points could be labeled in 2^n distinct ways. Our finite class C of functions will consist of 2^n "templates" that cover every possible labeling of points. As an example, consider when $n = 2$. We could have points in X labeled: $[+, +], [-, -], [-, +], [+, -]$. Our finite class of functions C would also be: $[+, +], [-, -], [-, +], [+, -]$. In assigning labels using our function, we would simply consider the points $x_1, x_2, \ldots x_n$ in ascending order and match each point with the label in our selected function like: $x_1 : c_1, x_2 : c_2, \ldots, x_n : c_n$.

As demonstrated in 3.a, with this particular class C , consisting of 2^n members, our VC dimension is $log(2^n) = n$, because that is how many points we could effectively cover. As soon as we have $n + 1$ points, we would need 2^{n+1} hypothesis functions to make sure we cover all possibilities.

4 Extra Credit - Decision Lists

1. Show that if a concept c can be represented as a k-decision list so can its complement, ¬c. You can show this by providing a k-decision list that represents ¬c, given c = {(c1,b1),...,(cl,bl),b).

If a concept c can be represented as a k-decision list, then its complement $\neg c$ can also be represented by simply flipping all of the b's. Say we have a list: $c = \{(x1, 1), (x2, 0), 1\}$. We see that if

$$x1, x2 = [1, 0], c = 1$$
$$x1, x2 = [0, 1], c = 0$$
$$x1, x2 = [0, 0], c = 1$$
$$x1, x2 = [1, 1], c = 1$$

$c's$ complement would have an output here = 0,1,0,0 given the same examples. If we simply switch all of the $b's$ from 0 to 1 and vice versa ($\neg c = \{(x1, 0), (x2, 1), 0\}$) we see that that is exactly our output.

2. Use Occam's Razor to show: For any constant k ≥ 1, the class of k-decision lists is PAC-learnable.

To show that the class of k-decision lists is PAC-learnable, we need to identify the size of the class $|H|$. To do this, I'll proceed similarly to how we figured out the size of the class of 3-CNF's in class. Let's first consider how many unique k literals we have. We have a set of n variables that we can choose to use in each slot of our conjuncts, and each of these variables can either be: x_n or $\neg x_n$, meaning we have a total of $2n$ variables to choose from in our "bucket". If each conjunct was exactly k literals, then our total number of conjuncts would be $\binom{2n}{k}$, but we have at most k, thus the number of unique k literals is:

$$\binom{2n}{k} + \binom{2n}{k-1} + \cdots + \binom{2n}{0}$$

Which is $O((2n)^k)$. Then, for each literal, we could either label it 0, label it 1, or not include it in our list. This means that the # of potential lists would be $O(3^{2n^k})$.

For PAC learnability, the quantity of interest is $log(|H|)$. This quantity needs to be polynomial in n in order for PAC learnability. When we plug in our answer $|H| = O(3^{2n^k})$, we get:

$$\begin{aligned} log(3^{2n^k}) &= 2n^k log(3) \\ &= O(n^k) \end{aligned}$$

This is polynomial in n, thus, the class of k-decision lists is PAC-learnable.

3. Show that 1-decision lists are a linearly separable functions. (Hint: Find a weight vector that will make the same predictions a given 1-decision list.)

A 1-decision list consists of single variables as the conjucts in the decisions list. For this class to be linearly separable, every such decision list would need to be able to be represented by a linear equation that shares the same predictions as the list. In order for this to be the case, one potential way to construct the weights would be to make the weights of $+$ predictions = factorial(length of list - index in list), and the $-$ predictions = - factorial(length of list - index in list).

For instance, say we are given the list $c = \{(x1, 1), (x2, 0), (x3, 0), 0\}$. Our corresponding weight vector would be $W = [3!, -2!, -1!], b = -1$. If our example $x = [1, 1, 1]$ we see that with this decision list, we should see a positive prediction. Here is the linear classifier prediction:

$$\begin{gathered} x_1(w_1) + x_2(w_2) + x_3(w_3) + b \geq 0 \\ (1)(3!) + (1)(-2!) + (1)(-1!) + -1 \geq 0 \\ 6 + -2 + -1 + -1 \geq 0 \end{gathered}$$

Which is true, thus returning a positive prediction. I chose to use factorials here as a way to ensure that conjunctions which are satisfied early in the list have their predictions dominate no matter what comes after them.