Machine learning 2 - Final Assignment

Frans de Boer
fransdeboer
5661439

June 30, 2022

The git repository for this assignment (including code and the report) can be found at github.com. I will make this repository public after the Brightspace deadline has passed.

1 PAC Learning

1.a Extend the proof that was given in the slides for the PAC-learnability of hyper-rectangles: show that axis-aligned hyper-rectangles in n-dimensional feature spaces (n > 2) are PAC learnable.

We can extend the proof by taking the number of sides/faces of the hyper-rectangle into account.

- A hyper-rectangle in 2-dimensional space is simply a rectangle and has 4 sides.
- A hyper-rectangle in 3-dimensional space is a rectangular box and has 6 sides.

For now I will assume the number of sides on a n-dimensional hyper-rectangle to be $F(n)$, and I will get back later to the calculation of $F(n)$.

To start we can follow the steps from the lecture. We have the ground truth n-dimensional hyper-rectangle R, and the current tightest fit rectangle R'. The error $(R - R')$ can be split into $F(n)$ strips T'. On each of these strips we can now grow a new strip T such that the probability mass of T is $\frac{\varepsilon}{F(n)}$.

If T covers T' for all strips then the probability of an error is

$$P[error] \leq \sum_{i=0}^{F(n)-1} P[T_i] = F(n)\frac{\varepsilon}{F(n)} = \varepsilon \tag{1}$$

We can now estimate the probability that T does not cover T'

$$P[\textit{random x hits T}] = \frac{\varepsilon}{F(n)}$$
$$P[\textit{random x misses T}] = 1 - \frac{\varepsilon}{F(n)}$$
$$P[\textit{m random x's misses T}] = (1 - \frac{\varepsilon}{F(n)})^m$$

Since we have $F(n)$ strips:

$$P[\textit{m random x's miss any Ts}] \leq F(n)(1 - \frac{\varepsilon}{F(n)})^m$$
$$P[\textit{R' has larger error than } \varepsilon] \leq F(n)(1 - \frac{\varepsilon}{F(n)})^m \ < \delta$$

Bounding the chance that our R' has an error larger than ε by δ:

$$F(n)(1 - \frac{\varepsilon}{F(n)})^m \ < \delta$$

Using $e^{-x} \geq (1-x)$

$$F(n)e^{-m\varepsilon/F(n)} \geq F(n)(1 - \frac{\varepsilon}{F(n)})^m$$

So instead we can use:

$$F(n)e^{-m\varepsilon/F(n)} < \delta$$
$$-m\varepsilon/F(n) < \log(\delta/F(n))$$
$$m\varepsilon/F(n) > \log(F(n)/\delta)$$
$$m > (F(n)/\varepsilon)\log(F(n)/\delta)$$

So any n-dimensional hyper-rectangle is learnable.

1.b Assume we have a 2-dimensional feature space R2, and consider the set of concepts that are L1-balls: $c = \{(x,y) : |x| + |y| \leq r\}$ (basically, all L1-balls centered around the origin). Use a learner that fits the tightest ball. Show that this class is PAC-learnable from training data of size m.

Same thing? what. literally just question a but now rotated 45 degrees

1.c Now we extend the previous class by allowing the varying center: $c = \{(x,y) : |x-x0| + |y-y0| \leq r\}$. Is this class PAC-learnable? Proof if it is, or otherwise disproof it.

maybe, maybe not. can't you still just do the tightest fit L1 ball? consistent learner so PAC learnable?

1.d Generate data according to, what you would argue is, the 'best case' scenario, i.e., for which learning is easiest. Clearly specify how you generate this data and show the resulting learning curve.

First I'll discuss the setup I'm using to run these tests. I'm generting L_1 balls with $r = 1$. I've set the size of the sample space to 2. This means that the sample space is a $2x2$ square with the center at $(0,0)$. The surface area of this square is 4, and the surface area of the L_1 ball is 2, exactly half of the sample space.

For each tests I'll be using a sample size of $m \in \{2,...,10000\}$, and I'll be taking the average of 100 trials. From a set of samples m the surface area of the L_1 ball can be determined, and from here the error of the ground truth can be calculated. This error is

$$(2 - estimated\_area)/2$$

2 is the surface area of the ground truth L_1 ball. This is divided by 2 because only positive samples that fall into this area will be misclassified (negative samples are never misclassified).

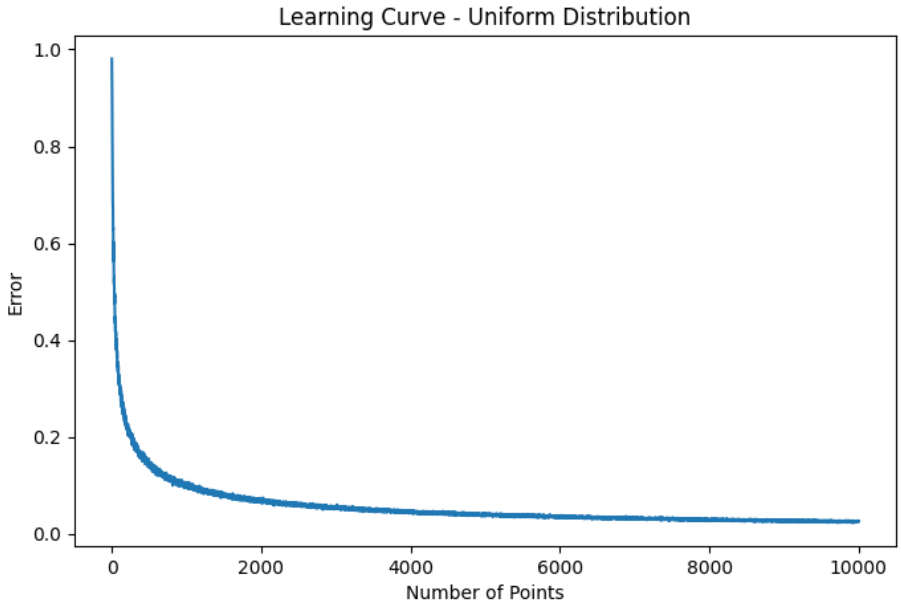
First I wanted to run a test using a basic uniform distribution to confirm the code is working properly. The results from this test can be seen in Figure 1. The more samples get added, the lower the error will be.

I personally think the best case scenario would be a method that generates samples close to the edge of the L_1 ball. To do this I generated values for polar coordinates and converted these to cartesian coordinates in the following way:

$$\begin{aligned} r_i &\; N(1, 0.2) \\ d_i &\; U(0, 2\pi) \\ x_i &= r_i \cos(d_i) \\ y_i &= r_i \sin(d_i) \end{aligned}$$

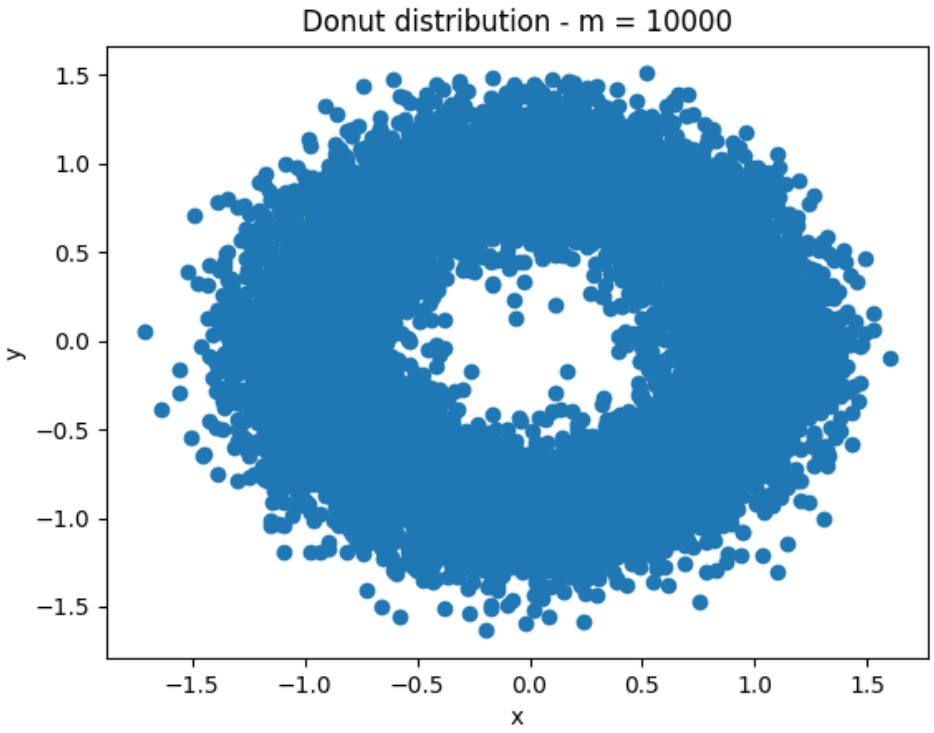
This will generate samples following a donut like distribution (see Figure 2 for an example of generated points). The learning curve can be seen in Figure

Figure 1: Learning Curve for a points sampled from a uniform distribution



1.e Similarly, generate data according to, what you would argue is, the 'worst case' scenario. Clearly specify how you generate this data and show the resulting learning curve. How close is this curve to the theoretical PAC bound?

Figure 2: Donut Distribution ($m = 10000$)



2 VC Dimension

Let us assume we are dealing with a two-class classification problem in d-dimensions. Let us start off with refreshing our memories. Consider the class of linear hypotheses (i.e., all possible linear classifiers) in d-dimensional space.

2.a Given N data points, how many possible labelings does this data set have?

Each data point can have 2 labels, so 2^N

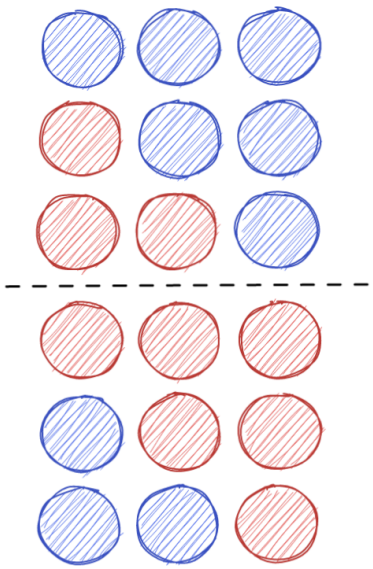
2.b What is the dimensionality d we need, at the least, for our class of linear hypotheses to be able to find perfect solutions to all possible labelings of a data set of size N? Stated differently, what is the smallest d that allows us to shatter N points?

$d-1$

2.c Consider d = 1 and a data set of size N. At maximum, how many different labelings of such a data set can decision stumps solve perfectly, i.e., with zero training error?

This would be all cases where there is no overlap, i.e. all points with class 1 (or 2) are either all on the left or right side. See Figure 3 for an example with N=3. Note that the first 3 and last 3 examples are the same, just with inverted labelings.

Figure 3: Example of perfectly solvable labels with $N = 3$



With a dataset of size N, there are N ways in which we can have class 1 be on the left side and have a decision stump be able to solve it perfectly. There can be $1, 2, ..., N$ points with class 1 in a row. Because each of these possible label assignments has a compliment (just invert the classes), we end up with a total of $2N$ possible labelings that a decision stump can solve perfectly.

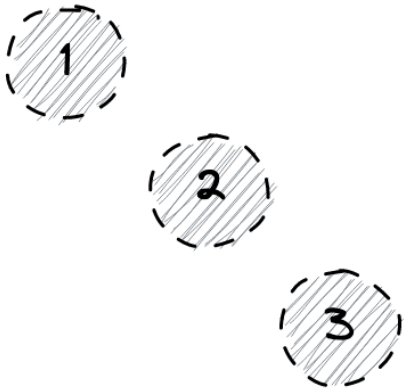
2.d With the answer from 3.c, give an upper-bound on the maximum number of different labelings a data set of size N in d dimensions can get by means of decision stumps.

We can once again look at Figure 3, but now instead of true labels we can view the labels (red and blue) as labels predicted by the decision stump. The decision stump either assigns all points to the left of it as class 1 or as class 2. So in $d = 1$ we know the maximum number of different labelings a data set of size N can get is $2N$.

This can be viewed in another way, for every dimension the decision stump can make $2N$ different labelings. For example in 2D (Figure 4) the decision stump for the x dimension can make $2N$ labelings, and the decision stump for the y dimension can make $2N$ labelings.

This continues for each dimension added, so we end up with a total of $2Nd$ maximum number of different labelings a data set of size N in d dimension can get by means of decision stumps.

Figure 4: Example with $d = 2$ and $N = 3$



I'd like to add some more notes to this solution. This is not neccesary to read, but it sheds some light on the work I did for this question and some of the mistakes I made.

While trying to answer this question I pretty quickly came to the answer of $2Nd$, but then later thought it to be wrong because this formula does not take into account duplicate labelings. In Figure 4 for example the x and y decision stumps can create many duplicate label assignments.

I tried to find patterns in combinations of N and d, for example I noticed that for $d = 2$ only $N \leq 3$ was solvable (see Figure 5), and for $d = 3$ at least $N \leq 4$ was solvable (see Figure 6, here smaller circles can be seen as being further away in the z dimension).

From here it seemed like there was a pattern where for any $N \leq (d+1)$ the problem was fully solvable, so the number of possible labelings the decision stump could make was 2^N. I could however not find a pattern in the number of labelings a decision stump could make for problems with $N > (d+1)$.

Figure 5: Solvable example with $d = 2$ and $N = 3$

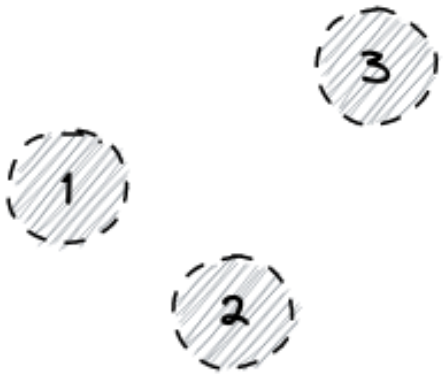
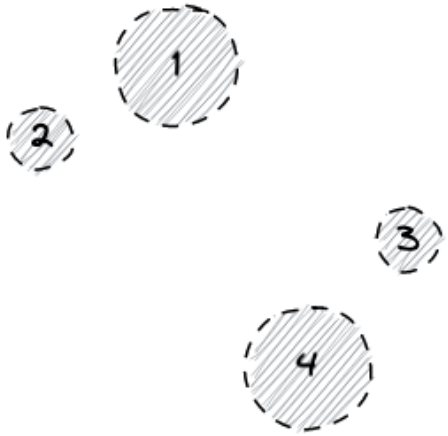


Figure 6: Solvable example with $d = 3$ and $N = 4$



It was not until a day of thinking and researching that I found these Slides (Page 5) that confirmed my original answer.

Dimensionality	Upper-bound on VC-Dimension
1	2
2	4
3	4
4	5
5	5
6	6
7	6
8	6
9	6
10	7
11	7
12	7
1024	14
2^{100}	107

Table 1: Results for upper bounds on the VC Dimension

2.e Using the bound from Exercise 3.d, determine the smallest upper-bound on the VC-dimension for decision stumps for dimensionalities $d \in \{1,2,3,4,5,6,7,8,1024,2^{100}\}$

I used the definition from the book *Foundations of Machine Learning* [1] for the upper-bound on the VC-Dimension

Smallest upper-bound on VC-dimension: To give an upper bound, we need to prove that no set S of cardinality N can be shattered

Since we know the maximum number of different labelings a data set of size N in d dimensions can get by means of decision stumps to be $2Nd$ (Section 2.d), and the total number of possible different labelings of a data set of size N can get is 2^N (Section 2.a), we have to find the first case where $2Nd < 2^N$. In other words, find the first N for which the classifier cannot create every possible labeling.

Since I could not find an analytical solution to this I decided to solve it programmatically. The results can be seen in Table 2. I've also added results for $d \in \{9,10,11,12\}$ for Section 2.g.

To me one interesting thing about these results is that they did not seem to agree with some other results I had found online. These slides mentions the growth should be proportional to $\log(d)$, which it is. But This homework assignment gives an upper bound on the VC dimension of $2(\log_2(d)+1)$.

Dimensionality	Lower-bound on VC-Dimension
1	**2**
2	3
3	**4**
4	4
5	**5**
6	5
7	5
8	5
9	5
10	5
11	5
12	6

Table 2: Results for experimentally found lower bounds on the VC Dimension

2.f Write a program that finds lower-bounds experimentally and that would even find the true VC-dimension given enough time (and a sufficient numerical precision). Provide a clear description for (possibly in terms of pseudo code) and reasoning behind the program. (Hint: given an arbitrary data set of of size N, you can of course always check if the class of decision stumps shatters it.)

Once again the definition from the book *Foundations of Machine Learning* [1] for the lower-bound on the VC-Dimension is used

Upper-bound on VC-dimension: To give a lower bound, we need to show that a set S of cardinality N can be shattered

2.g Determine for every dimensionality $d \in \{1,2,3,...,10,11,12\}$ the true VC-dimension and proof that, indeed, it is the actual VC-dimension for decision stump in spaces of that dimensionality. (Note: please be advised that you are not expected to completely crack this open(?) problem, but do have a go at it and see how far you can get in a reasonable amount of time. Hint: you probably can get some inspiration from the lower-bounds your program from 3.f finds.)

References

[1] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2 edition, 2018.