Machine Learning B
Home Assignment 3

Jakob Schauser, pwn274

December 2021

1 The VC-dimension

1.1

We know the definition of the VC-dimension:

$$d_{\mathrm{VC}}(\mathcal{H}) = \max\{n \mid m_{\mathcal{H}}(n) = 2^n\} \tag{1}$$

Last time we proved

$$m_{\mathcal{H}}(n) \leq min(M, 2^n) \tag{2}$$

Combining these we can see that n cannot be above $log_2(M)$ for $m_{\mathcal{H}}$ to be 2^n (as is said in equation (2)). This means that n in equation (1) will be:

$$d_{\mathrm{VC}} \leq log_2(M) \tag{3}$$

as for any n above this will leave the growth function too small to shatter, which makes sense, as 2^n hypotheses are needed to shatter n points.

1.2

Given two hypotheses, for one point the possiblilities are: (+) and (-). This is of course shattered by two hypotheses. Moving up to two points, we have the possibilities (+,+), (+,-), (-,+) and (-,-). Only given two hypotheses, these options cannot be exhausted. It is therefore trivial to see that it is only for 1 point that two hypotheses can shatter, which, by the definition of the d_{VC} gives:

$$d_{VC} = 1 \tag{4}$$

1.3

I start out by looking at **Theorem 3.16**:

$$\mathbb{P}\left(\exists h \in \mathcal{H} : L(h) \geq \hat{L}(h,S) + \sqrt{\frac{8\ln\left(2\left((2n)^{d_{\text{VC}}}+1\right)/\delta\right)}{n}}\right) \leq \delta \tag{5}$$

I can now insert the given values in their respective places and and simplify:

$$1 \geq \sqrt{\frac{8\ln\left(2\left((2n)^{d_{\text{VC}}}+1\right)/\delta\right)}{n}} \Leftrightarrow n \geq 8\ln\left(2\left((2n)^{d_{\text{VC}}}+1\right)/\delta\right) \tag{6}$$

Now assuming large n, I choose to approximate in the following:

$$n \geq 8\ln\left((2n)^{d_{\text{VC}}}\right) = 8d_{\text{VC}}\ln(2n) \tag{7}$$

Now moving some terms around, I can find the approximate relation:

$$n/\ln(2n) \geq 8d_{\text{VC}} \tag{8}$$

1.4

Theorem 3.16

$$\mathbb{P}\left(\exists h \in \mathcal{H} : L(h) \geq \hat{L}(h,S) + \sqrt{\frac{8\ln\left(2\left((2n)^{d_{\text{VC}}}+1\right)/\delta\right)}{n}}\right) \leq \delta \tag{9}$$

Theorem 3.2

$$\mathbb{P}\left(\exists h \in \mathcal{H} : L(h) \geq \hat{L}(h,S) + \sqrt{\frac{\ln\frac{M}{\delta}}{2n}}\right) \leq \delta \tag{10}$$

Isolating the two bounding factors and simplifying gives:

$$8\ln\left(2\left((2n)^{d_{\text{VC}}}+1\right)/\delta\right) <> \ln\frac{M}{\delta}/2 \tag{11}$$

Now using what I learned in (1.1), namely $d_{\text{VC}} \leq log_2(M)$:

$$8\ln\left(2(2n)^{\leq log_2(M)}+2\right) <> \ln M/2 \tag{12}$$

Exponentiating both sides gives:

$$2e^8\left((2n)^{\leq log_2(M)}+1\right) <> M/2 \tag{13}$$

Now we can see, that only the left side depends on n, meaning that for a large number of data points the right bound (**Theorem 3.2**) is tighter. From what I can gather this is also the case for large M, unless we have a very small number of data points.

1.5

Looking at **Theorem 3.16** for a linear separator in R^{10}, we have:

$$\mathbb{P}\left(\exists h \in \mathcal{H} : L(h) \geq \hat{L}(h, S) + \sqrt{\frac{8\ln\left(2\left((2n)^{11}\right)/\delta\right)}{n}}\right) \leq \delta \tag{14}$$

Inserting the confidence:

$$0.01 \geq \sqrt{\frac{8\ln\left(2\left((2n)^{11}\right)/0.01\right)}{n}} \tag{15}$$

Plugging this into Wolfram Alpha gave the following solution:

$$n \geq 1.561 \times 10^7 \tag{16}$$

Meaning a lot of data is needed.

1.6

`https://www.tiktok.com/@jakobschauser/video/7041529517453724934?lang=en&is_copy_url=0&is_from_webapp=v1&sender_device=pc&sender_web_id=6915352695297787397`

$$\text{QED} \ \square \tag{17}$$

1.7

In the last question, a single circle 'hugging' four points in a square would not be able to reach two in a diagonal without also 'hugging' one of the other corners.

Now, image the three points in the same configuration as last, but with a fourth point some distance away:
When only using the positive circles, the three points can be 'shattered' as shown in 1.6, but the fourth is only ever negative as it is never within any of the circles. Now doing the same for the negative circles, we can get all the same configurations but with the fourth point flipped to positive. It is now easy to see, that this will shatter all four points, as all possibilities are exhausted, so we have once again found a lower bound by example.

1.8

This problem was not

1.9

As each leaf in the final layer of the tree corresponds to a binary classification, we can map each leaf to a point. As all combinations of leafs are possible, the maximal number of points that will be shattered must equal the number of leafs. I assume we are looking at symmetric decision trees. As this is the exact definition of the VC-dimension, we can say that:

$$d_{VC} = 2^d \tag{18}$$

1.10

Now, given a large enough tree, we can have any number of leafs, meaning any number of points can be shattered, giving, per definition:

$$d_{VC} = \infty \tag{19}$$

1.11

Theorem 3.21 is an obvious place to start:

$$\mathbb{P}\left(\exists h \in \mathcal{H}_\gamma : L_{\text{FAT}}(h) \geq \hat{L}_{\text{FAT}}(h,S) + \sqrt{\frac{8\ln\left(2\left((2n)^{d_{\text{FAT}}(\mathcal{H}_\gamma)}+1\right)/\delta\right)}{n}}\right) \leq \delta \tag{20}$$

I can find a bound on the d_{FAT} by the following relation:

$$d_{\text{FAT}}\left(\mathcal{H}_\gamma\right) \leq \left\lceil R^2/\gamma^2 \right\rceil + 1 \tag{21}$$

As we are exactly looking on inputs within the unit ball, and we are given a margin on 0.1, this is:

$$d_{\text{FAT}}\left(\mathcal{H}_\gamma\right) \leq \left\lceil 1^2/(0.1)^2 \right\rceil + 1 = \lceil 100 \rceil + 1 = 101 \tag{22}$$

Once again, inserting our certainties and numbers, the correction term boils down to:

$$\sqrt{\frac{8\ln\left(2\left((2\cdot 10^5)^{\leq 101}+1\right)/0.99\right)}{10^5}} \approx (\leq 0.1044) \tag{23}$$

This means that with a probability of 99%:

$$\exists h \in \mathcal{H}_\gamma : L_{\text{FAT}}(h) \geq \hat{L}_{\text{FAT}}(h,S) + (\leq 0.1044) \tag{24}$$

...

After speaking with my peers when grabbing a cup of coffee, I realize I might have used the wrong bound, as the exponent is also an upper bound. I believe I should have used **Theorem 3.22**, but I do not have the time to change it now.

1.12

I did not have time to solve this.

2 The t-SNE Algorithm

2.1

The t-SNE was implemented, as it was quite slow when I did my first tests, it was made almost entirely loopless, taking advantage of a bunch of Numpy-functionality.

The distance matrix was found using the following two imported SciPy-functions:

```
dists = squareform(pdist(Ys))**2
```

Given a sigma-array, the affinities could then be found this way:

```
top = np.exp(-dists/(2*sigma**2))

np.fill_diagonal(top,0)

bottom = np.sum(top,axis = 0)

affinities = top/bottom
```

One the affinities were found, the perplexities were calculated. Here, the zeros on the diagonal lead to some -inf's, these were taken care of:

```
logs = np.nan_to_num(np.log2(affinities))

perplex = np.power(2,np.sum(-affinities*logs,axis = 0))
```

Now what was needed was the sigmas. These needed to be found by a binary search for a chosen perplexity. The search was also vectorized and implemented in the following way:

```
perp = get_perplex(sigmas)

above = perp > perplex_goal
below = perp < perplex_goal

sigma_max[above] = sigmas[above]
sigma_min[below] = sigmas[below]

sigmas = (sigma_max + sigma_min)/2
```

I was not very creative and choose a perplexity of 50 as was mentioned in the paper. The code finds the sigmas in about 20 iterations/55 seconds.

Now the actual t-SNE gradient descend was coded. I started out by writing the implementation as it was described in the paper, and then spend quite a lot of time trying to vectorize it. This snippet was harder to clean for easy reading, but I hope it is legible what happens:

```
# I break the derivative up into smaller terms
p_minus_q = affinities - low_dim_affinities

# Making a three dimensional difference of every Y
difference_between_ys = Ys[None,:,:] - Ys[:,None,:]

# Some clever dimension manipulation so the
# sum works in the later steps
left = np.dstack((p_minus_q,p_minus_q))*difference_between_ys

# Finding the distances
right = 1+squareform(pdist(Ys))**2

# The derivative can now be found in a single numpy-vector calculation
dCdy = 4 * np.sum(left/np.dstack((right,right)), axis = 0)

# Updating the Ys:
Ys += -eta*dCdy + alpha(t)*(last_Ys-second_Ys)
```

2.2

We start out by plotting the original data:

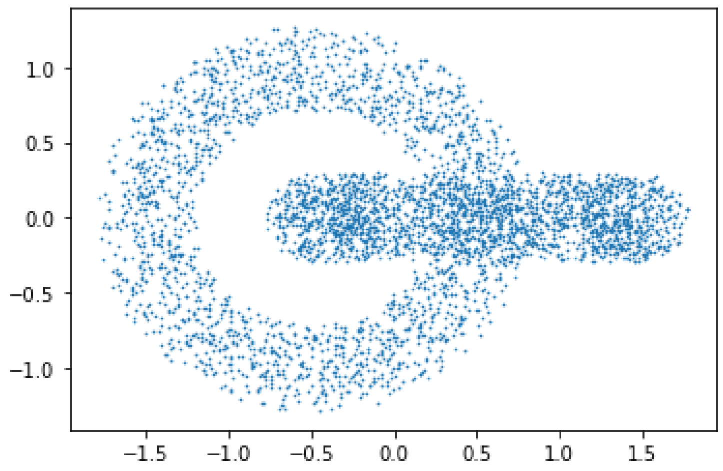

Figure 1: The original data

Here, it is clear to see, that the data is two interlinked toruses. If the two tori were of two different classes, it can be seen that it would be highly non-trivial to separate in the current dimentionality.

A person showing sublime naïveté might attempt a PCA,

Doing the PCA was quite a lot easier, as I simply implemented the SciKitLearn package and did the transformation in the following three lines using only the standard parameters:

```
from sklearn.decomposition import PCA
pca = PCA()
trans = pca.fit_transform(points)
```

The data, when transformed by PCA looks as follows:

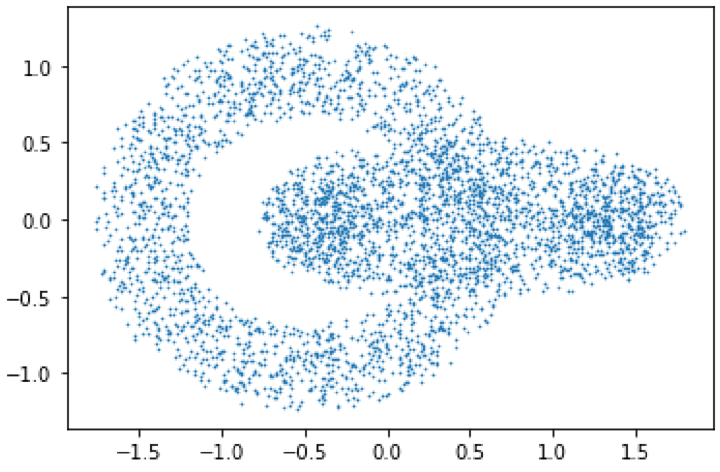

Figure 2: The PCA transformed data view in the two most

This is basically the same as the original data, rotated as to have a slightly higher variance. Once again, clean separation seem impossible.

Now I implement my PCA, using a Learning Rate (eta) of 0.1, a momentum (alpha) of 0.1.

I let it run for about 200 epochs which took just above 9 minutes.

The results after 0, 80 and 200 epochs are as follows:

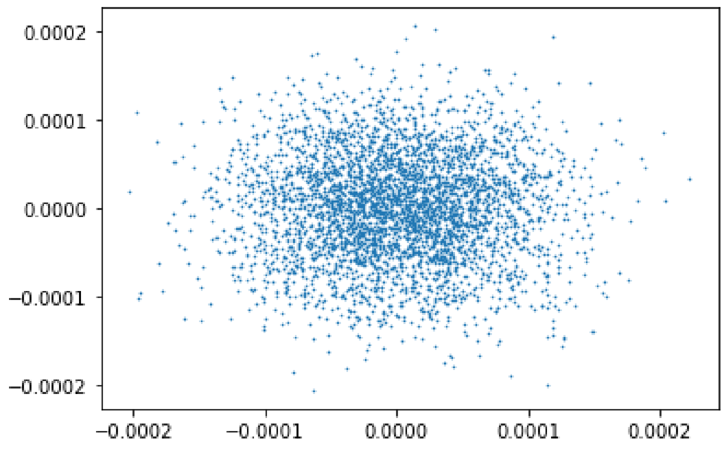

Figure 3: After 0 iterations

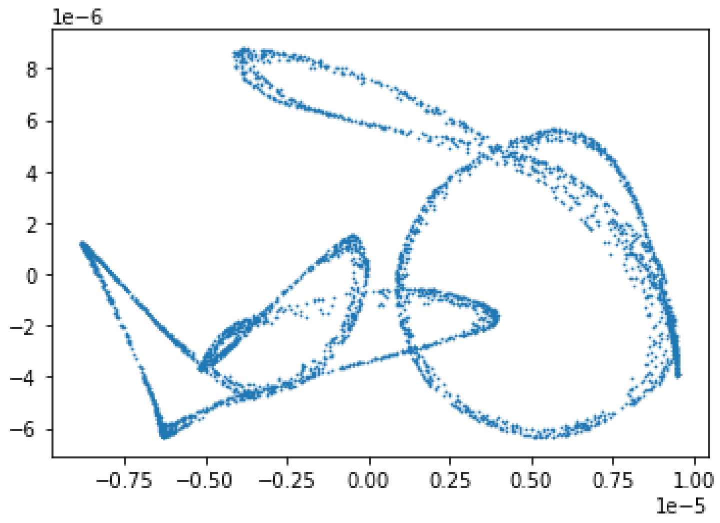

Figure 4: After 80 iterations

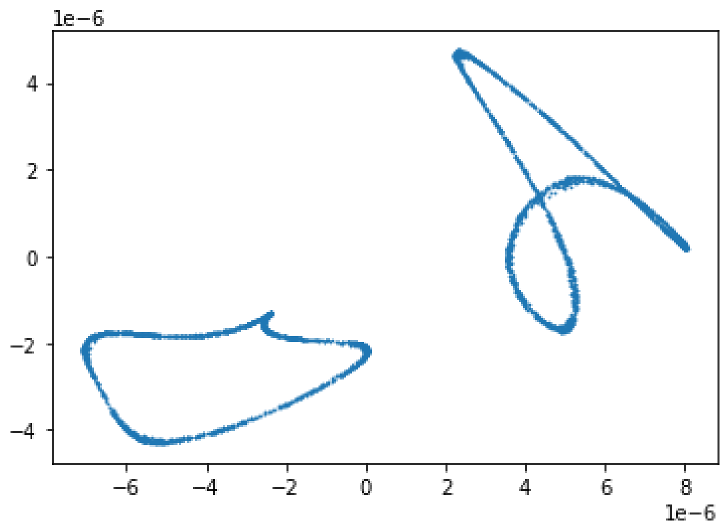

Figure 5: After 200 iterations

It is clear that some structure, and two different groupings are found. It can also be seen that, in some high dimensional space the two groups might have a local ring-shape, as the left groups is almost circular and the right looks like a bend torus that is rotated in a third dimension. I believe, that given more iterations, the data would have resettled into two circles.

2.3

I already talked about this in the preceding questions, but to recap:

The original data has two tori that are linked, making the two data-clumps quite literally inseparable.
Simply using a PCA does not help with this problem, as no simple projection of the data could allow for a clean split.
After running the t-SNE, the data separated into the two obvious subgroupings i.e. the two interlocked rings.