

HTML HW1
B12901022 廖冠豪

5

The answer by the agent is as follows

Active learning is a machine learning approach where the model is trained interactively by selecting the most informative data points from which to learn, rather than being trained on a large set of randomly selected data. This can be particularly useful when labeled data is scarce or expensive to obtain.

One possible application of active learning is in medical image classification. Medical experts' time is expensive, and annotating images like X-rays, MRIs, or CT scans can be labor-intensive. In an active learning scenario, a machine learning model would initially be trained on a small labeled dataset. As the model identifies new cases where it is uncertain, it queries the experts to label these specific cases. Over time, the model improves its performance while minimizing the number of images that need manual labeling, leading to cost and time efficiency.

Other applications of active learning include:

- Natural language processing tasks (e.g., text classification, sentiment analysis)
- Autonomous driving (e.g., labeling road scenes with uncertain or rare conditions)
- Fraud detection in financial systems

I agree mostly with the agent's answer. Active learning is suitable for cases in which unlabeled data is abundant but manual labeling is expensive. All the answers provided by the agent fit this criteria. Unlabeled data is abundant in these cases, but labeling requires human expertise, which can be costly. The agent further explains how active learning can be used to tackle these cases. The active learning model asks the human expert when encountering a data case of which it is unsure. Its performance improves during the process and the number of queries to human experts is minimized. The one thing I don't like about the agent's answer is that it doesn't provide any real-life examples in which active learning is actually applied to any of the cases it mentioned. This is a flaw to me because when being asked what "applications" active learning has the expected answer should be fields in which active learning is actually used and contributing, instead of cases that theoretically would be a suitable problem for active learning. In conclusion, as the agent's boss, I'm mostly satisfied with the answer it provides, but this satisfaction is compensated by the lack of real-life example in its answer.

6

The answer by the agent is as follows(some content that isn't directly related to the question is omitted)

While machine learning (ML) has potential in predicting aspects of earthquake activity, it is still not fully reliable for precise earthquake predictions. The unpredictable nature of seismic events, combined with the complexity of underlying geological processes, makes earthquake prediction a highly challenging task.

Challenges

- Lack of sufficient labeled data: While seismic data is abundant, labeled data for events directly leading to earthquakes is rare and often ambiguous.
- Complexity of Earth systems: The geological processes underlying earthquakes are extremely complex and not fully understood, limiting the ability of even the most advanced models.

I agree with the agent's answer. The difficulties in trying to precisely predict earthquake events is clearly marked in the answer. First, it's hard to obtain well-labeled data for the model to train from, as the causes of earthquakes is still not fully known to human experts. Therefore, the reliability of the data cannot be guaranteed. As we've learned from the lecture, the No Free Lunch Theorem states that no meaningful result can be produced by learning without correct assumption. Given that scientists have not a complete understanding of the mechanism behind earthquakes, it's hard to make the "correct assumption". Second, given the complex geological nature, all earthquakes might not share a common cause. It might be that multiple different geological phenomena can all lead to earthquakes. If this is the case, it would be hard to train a model to predict earthquakes by recognizing underlying common patterns, as there could be none. In conclusion, as the agent's boss, I agree with and am satisfied with the answer provided, as the argument is clear and fits my understanding of geology.

7

In this problem, we set $\text{sign}(0) = 1$.

Consider the case where

$$N = 1, \mathbf{x}_1^{\text{orig}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, y_1 = -1$$

We first run PLA on $(\mathbf{x}_n, y_n)_{n=1}^1$. Since $\text{sign}(\mathbf{w}_0^T \mathbf{x}_1) = 1 \neq y_1$, \mathbf{w} is updated by

$$\begin{aligned} \mathbf{w}_1 &\leftarrow \mathbf{w}_0 + y_1 \mathbf{x}_1 \\ \Rightarrow \mathbf{w}_1 &= \begin{bmatrix} -2 \\ -1 \\ -1 \end{bmatrix} \end{aligned}$$

After this update, there are no more mistakes, so the algorithm returns

$$\mathbf{w}_{\text{PLA}} = \begin{bmatrix} -2 \\ -1 \\ -1 \end{bmatrix}$$

We then run PLA on $(\mathbf{x}'_n, y_n)_{n=1}^1$. Since $\text{sign}(\mathbf{w}_0^T \mathbf{x}'_1) = 1 \neq y_1$, \mathbf{w} is updated by

$$\begin{aligned} \mathbf{w}_1 &\leftarrow \mathbf{w}_0 + y_1 \mathbf{x}'_1 \\ \Rightarrow \mathbf{w}_1 &= \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \end{aligned}$$

After this update, there are no more mistakes, so the algorithm returns

$$\mathbf{w}'_{\text{PLA}} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

We then consider the example in \mathbb{R}^2

$$\mathbf{x}_{\text{ex}}^{\text{orig}} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

We see that

$$\begin{aligned}\text{sign}(\mathbf{w}_{\text{PLA}}^T \mathbf{x}_{\text{ex}}) &= \text{sign}(-1 + 1 + 0) = 1 \\ \text{sign}(\mathbf{w}'_{\text{PLA}}^T \mathbf{x}_{\text{ex}}) &= \text{sign}(-2 + 1 + 0) = -1\end{aligned}$$

Hence \mathbf{w}_{PLA} and \mathbf{w}'_{PLA} don't always return the same binary classification for examples $\mathbf{x}^{\text{orig}} \in \mathbb{R}^2$, and the statement is disproved.

8

In this problem, we set $\text{sign}(0) = 1$.

We first prove that **mistakes happen at the same values of t for the two processes** by mathematical induction. Let \mathbf{w}'_t and \mathbf{w}_t the weight vector at the t -th step of the two processes running on $(\mathbf{x}'_n, y_n)_{n=1}^N$ and $(\mathbf{x}_n, y_n)_{n=1}^N$ respectively.

At $t = 0$ $\mathbf{w}'_0 = \mathbf{w}_0 = \mathbf{0}$.

Suppose that from $t = 0$ to $t = t'$ all the mistakes (and updates) happen at $t \in \mathcal{T} = \{t_1, t_2, \dots, t_k\}$ for the two processes.

Hence we have

$$\begin{aligned}\mathbf{w}'_{t'} &= \sum_{t \in \mathcal{T}} y_t(3, 3\mathbf{x}_t^{\text{orig}}) \\ \mathbf{w}_{t'} &= \sum_{t \in \mathcal{T}} y_t(1, \mathbf{x}_t^{\text{orig}}) \\ \implies \mathbf{w}'_{t'} &= 3\mathbf{w}_{t'}\end{aligned}$$

And

$$\text{sign}(\mathbf{w}'_{t'}{}^T \mathbf{x}) = \text{sign}(\mathbf{w}_{t'}{}^T \mathbf{x}') \quad \forall \mathbf{x} = (1, \mathbf{x}^{\text{orig}}), \mathbf{x}' = (3, 3\mathbf{x}^{\text{orig}}), \mathbf{x}^{\text{orig}} \in \mathbb{R}^d$$

Therefore, the next test ($t = t' + 1$) is a mistake to either none or both of the processes, and the statement still holds.

By the analysis presented above, we now know that $\mathbf{w}'_{\text{PLA}} = 3\mathbf{w}_{\text{PLA}}$, and they return the same binary classification result for any $\mathbf{x}^{\text{orig}} \in \mathbb{R}^d$, and are equivalent.

9

Define the $d + 1$ -dimensional vector \mathbf{w}_f based on the dictionary by

$$\mathbf{w}_{f_0} = -3.5$$

$$\mathbf{w}_{f_i} = \begin{cases} 1, & \text{if the } i\text{-th word in the dictionary is more hatred-like} \\ -1, & \text{if the } i\text{-th word in the dictionary is less hatred-like} \end{cases} \quad \text{for } 1 \leq i \leq d$$

We have

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}_f^T \mathbf{x})$$

So \mathbf{w}_f is the perfect vector the PLA algorithm hopes to approximate.

We then consider the update process of \mathbf{w}

First we consider the inner product of \mathbf{w}_t and \mathbf{w}_f

$$\begin{aligned} \mathbf{w}_f^T \mathbf{w}_{t+1} &= \mathbf{w}_f^T (\mathbf{w}_t + y_{n(t)} \mathbf{w}_t) \\ &\geq \mathbf{w}_f^T \mathbf{w}_t + \min_n (y_n \mathbf{w}_f^T \mathbf{x}_n) \end{aligned}$$

Also,

$$\begin{aligned} y_n \mathbf{w}_f^T \mathbf{x}_n &= |f(\mathbf{x}_n)| \\ \implies \min_n (y_n \mathbf{w}_f^T \mathbf{x}_n) &= \min_n |z_+(\mathbf{x}_n) - z_-(\mathbf{x}_n) - 3.5| \geq \frac{1}{2} \quad (z_+(\mathbf{x}), z_-(\mathbf{x}) \in \mathbb{N}) \end{aligned}$$

Therefore starting from $\mathbf{w}_0 = \mathbf{0}$, after T updates(mistakes), we have

$$\mathbf{w}_f^T \mathbf{w}_T \geq \frac{T}{2}$$

Next, we consider the change of the length of \mathbf{w}_t .

$$\begin{aligned} \|\mathbf{w}_{t+1}\|^2 &= \|\mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}\|^2 \\ &= \|\mathbf{w}_t\|^2 + 2y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)} + \|\mathbf{x}_{n(t)}\|^2 \leq \|\mathbf{w}_t\|^2 + \max_n (\mathbf{x}_n^2) \end{aligned}$$

In this case we know that \mathbf{x} has $m + 1$ components that equal to 1 and others equal to 0, so $\|\mathbf{x}_n\|^2 = m + 1$

$$\implies \|\mathbf{w}_{t+1}\|^2 \leq \|\mathbf{w}_t\|^2 + (m + 1)$$

Therefore, starting from $\mathbf{w}_0 = \mathbf{0}$, after T updates we have

$$\|\mathbf{w}_T\|^2 \leq T(m + 1)$$

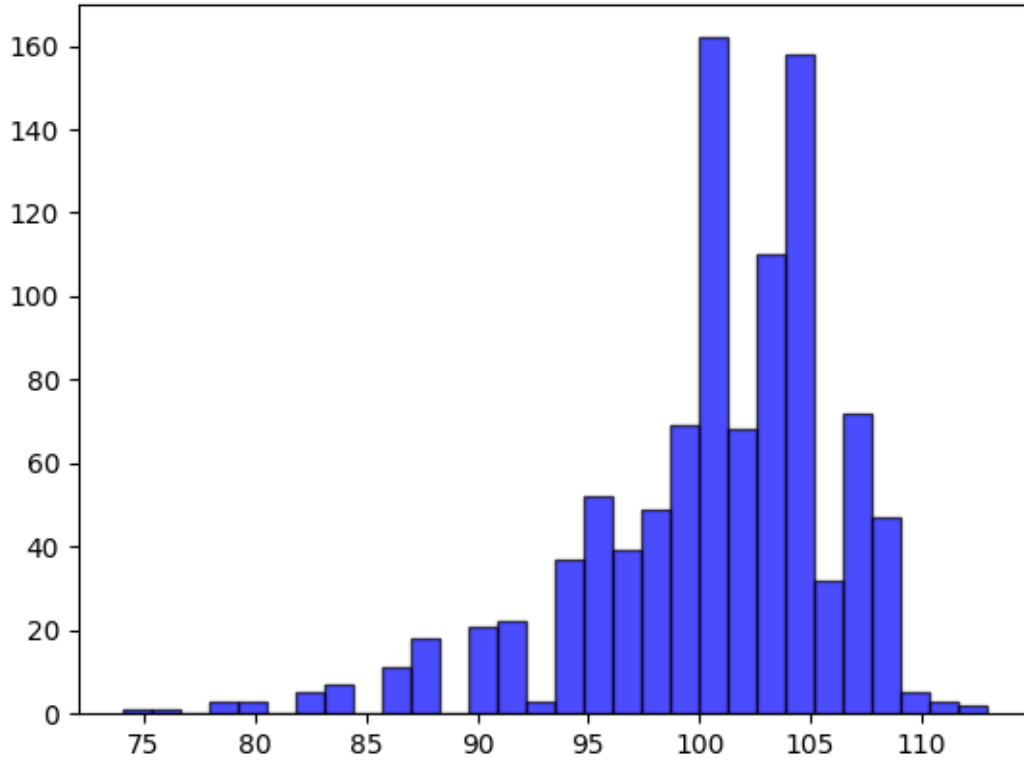
We finally consider the cosine of the angle between \mathbf{w}_f and \mathbf{w} after T updates

$$\begin{aligned} \frac{\mathbf{w}_f^T \mathbf{w}}{\|\mathbf{w}_f\| \|\mathbf{w}\|} &\geq \frac{T}{2\sqrt{T(m+1)} \|\mathbf{w}_f\|} \frac{1}{\|\mathbf{w}\|} \\ &= \frac{1}{2\|\mathbf{w}_f\|} \sqrt{\frac{T}{m+1}} \\ &= \sqrt{\frac{T}{4(m+1)(d+3.5^2)}} \end{aligned}$$

We also know that the cosine value has to be no greater than 1, so

$$\begin{aligned} \sqrt{\frac{T}{4(m+1)(d+3.5^2)}} &\leq 1 \\ \Rightarrow T &\leq (m+1)^2(4d^2 + 49) \end{aligned}$$

We hence obtain the upper bound for the number of mistakes this PLA algorithm can make.

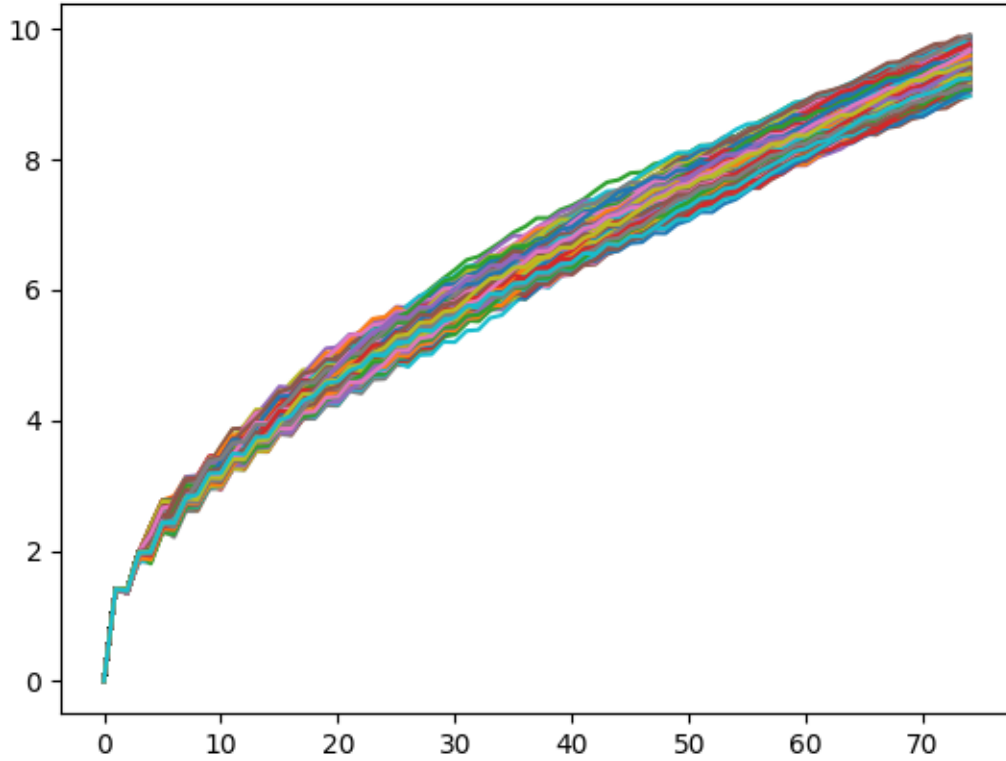


From the diagram presented above we see that throughout the 1000 experiments, the number of updates mainly lies between 100 and 105. The number of experiments with a total update count outside of this interval decreases rapidly as it gets further away from this interval. Also, we see that the decrease on the side $n < 100$ is quite smooth, with the minimal total update count of the experiments being around 75. On the other side, $n > 105$, the decrease is a lot more sharp, as the maximal total update count is no greater than 115. We can also see that the distribution of the total number of updates approximates a right-shifted normal distribution.


```
vim HW1.tex ~/D/2/H/HW1 (nvim)  %1  vim P12.cpp ~/D/2/H/HW1 (a.out)  %2  vim P10.cpp ~/D/2/H/HW1 (nvim)  %3  +
0 #include <cstdlib>
1 #include <fstream>
2 #include <vector>
3 #include <iostream>
4 using namespace std;
5
6 const int N = 200;
7 const int n = 47205;
8 int mnm = 1000;
9 ofstream data_out;
10
11 bool sign(double a){
12     if(a <= 0) return false;
13     return true;
14 }
15
16 class data_vector{
17 public:
18     bool y;
19     vector<pair<int, double> > x;
20
21     void init(string data_string){
22         y = true;
23         if(data_string[1] != ' '){
24             y = false;
25         }
26
27         vector<int> indexes;
28         for(int i = 0; i < data_string.length(); i++){
29             if(data_string[i] == ' ') indexes.push_back(i);
30         }
31         indexes.push_back(data_string.length());
32         x.push_back(make_pair(0, 1));
33     }
34 }
35
36 int main(){
37     data_vector dv;
38     dv.init(" ");
39     return 0;
40 }
```

NORMAL | ↩ main | > -4 | P10.cpp | utf-8 | cpp | Top | 1:1
"P10.cpp" 140L, 2724B written

Code snapshot

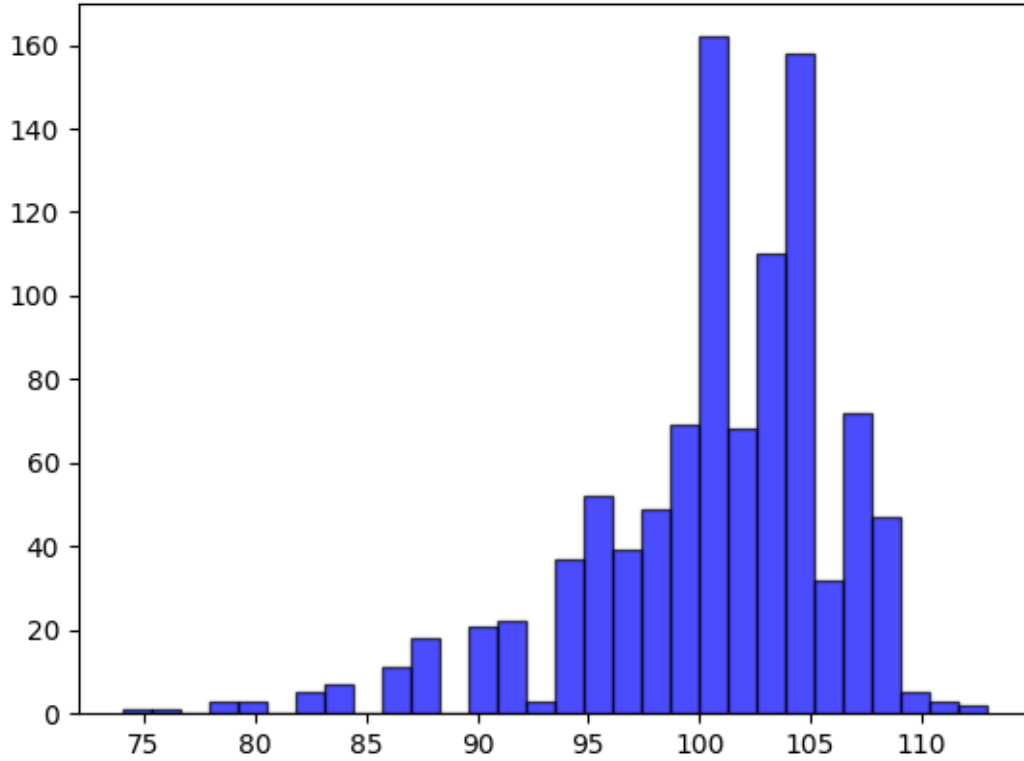


$$T_{min} = 74$$

From the diagram presented above we see that the length of \mathbf{w} is not always growing, with many small fluctuations during the process, but overall the length is increasing, starting from $\|\mathbf{w}_0\| = 0$ to $\|\mathbf{w}_{T_{min}}\| \approx 10$. Also, the growth of \mathbf{w} is a lot more rapid at the beginning of the process compared with the rest of the process.

```
vim HW1.tex ~/D/2/H/HW1 (nvim) 361 | vim P12.cpp ~/D/2/H/HW1 (a.out) 362 | vim P10.cpp ~/D/2/H/HW1 (nvim) 363 +
33 x.push_back(make_pair(0, 1));
32
31 for(int i = 0; i < indexes.size() - 1; i++){
30     string data_substr = data_string.substr(indexes[i], indexes[i + 1]);
29     int pos;
28     for(int j = 0; j < data_substr.length(); j++){
27         if(data_substr[j] == ':'){
26             pos = j;
25             break;
24         }
23     }
22
21     int index = stoi(data_substr.substr(0, pos));
20     double value = stod(data_substr.substr(pos + 1, data_substr.length()));
19     x.push_back(make_pair(index, value));
18 }
17 }
16
15 data_vector(){}
14 };
13
12 class weight_vector{
11 public:
10     vector<double> w;
9
8     weight_vector(){
7         w.assign(n + 1, 0);
6     }
5
4     bool update(data_vector example){
3         double inner_product = 0;
2         for(pair<int, double> p : example.x){
1             if(w[p.first] == 0) continue;
0             inner_product += w[p.first] * p.second;
NORMAL | 1 main | P11.cpp | utf-8 < [C] cpp | 44% | 67:1
"P10.cpp" 140L, 2724B written
```

Code snapshot



The median of number of updates is 101.

Comparison between Problem 10:

We run the 1000 experiments for 100 times, in every process the two PLA variants use the same random seed, but the seed is different among all processes. We compare the two PLA variants by comparing the median of number of updates in the 100 sets of experiments.

```

vim HW1.tex ~/D/2/H/HW1 (nvim)  #1 | vim P12.cpp ~/D/2/H/HW1 (a.out)  #2 | vim P10.cpp ~/D/2/H/HW1 (nvim)  #3 +
18 void train(int seed, vector<int> &v1, vector<int> &v2){
17     weight_vector w1;
16     weight_vector w2;
15     int count1 = 0;
14     int count2 = 0;
13     int update_count1 = 0;
12     int update_count2 = 0;
11
10     srand(seed);
9
8     while(count1 < 1000){
7         double random = double(rand()) / double(RAND_MAX);
6         int index = 199 * random + 1;
5         bool if_update = w1.update1(example_set[index]);
4
3         if(!if_update){
2             count1++;
1         }
0         else{
1             count1 = 0;
2             update_count1++;
3         }
4     }
5
6     while(count2 < 1000){
7         double random = double(rand()) / double(RAND_MAX);
8         int index = 199 * random + 1;
9         int update_times = w2.update1(example_set[index]);
10
11         if(update_times == 0){
12             count2++;
13         }
14         else{
15             count2 = 0;
16
NORMAL | main > +51 ~15 -4 P12.cpp utf-8 < c cpp 77% 159:1
W325: Ignoring swapfile from Nvim process 14821

```

Code snapshot

Denote the result of one set of experiments as (m_{10}, m_{12}) , where m_{10} is the median of number of updates when running the algorithm in P10 and m_{12} is that for P12.

The result is shown in the table below

(m_{10}, m_{12})	(101, 101)	(101, 102)	(102, 101)	(102, 102)
number of tests	68	13	18	1

We see that the difference between the two PLA variants is quite insignificant for this dataset. Nevertheless, the number of tests where the variant in P10 outperforms that in P12(judging by median of number of updates) is higher than that of the opposite case.

A possible explanation to the similarity between the performance of the two PLA variants might be that since $\mathbf{w}^T \mathbf{x}$ is usually relatively small in this case, most of the time one update is enough to make the new \mathbf{w} correctly classify the originally misclassified data point.

13

1

First we consider how \mathbf{w}_{t+1} classifies $\mathbf{x}_{n(t)}$

$$\begin{aligned}\mathbf{w}_{t+1}^T \mathbf{x}_{n(t)} &= \mathbf{w}_t^T \mathbf{x}_{n(t)} + \frac{1}{10} y_{n(t)} \|\mathbf{x}_{n(t)}\|^2 \left[\frac{-10 y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)}}{\|\mathbf{x}_{n(t)}\|^2} + 1 \right] \\ &> \mathbf{w}_t^T \mathbf{x}_{n(t)} + \frac{1}{10} \|\mathbf{x}_{n(t)}\|^2 \frac{-10 \mathbf{w}_t^T \mathbf{x}_{n(t)}}{\|\mathbf{x}_{n(t)}\|^2} = 0\end{aligned}$$

Hence $\text{sign}(y_{n(t)}) = \text{sign}(\mathbf{w}_{t+1}^T \mathbf{x}_{n(t)})$, and $\mathbf{x}_{n(t)}$ is correctly classified.

2

If the data is linearly separable, there exist a vector \mathbf{w}_f such that

$$\text{sign}(\mathbf{w}_f^T \mathbf{x}_n) = \text{sign}(y_n) \quad \forall (\mathbf{x}_n, y_n) \in \mathcal{D}$$

2.1

We first consider a different variant of PLA in which when $y_n \mathbf{w}_t^T \mathbf{x}_n \leq 0$ \mathbf{w}_t is updated by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \frac{1}{10} y_{n(t)} \mathbf{w}_{n(n)}$$

(The condition that $y_n \mathbf{w}_t^T \mathbf{x}_n \leq 0$ is almost equivalent to the condition that the example is falsely classified, but it alleviates us from the trouble of dealing with convention for $\text{sign}(0)$)

We then prove that this variant of PLA halts to a perfect hyperplane if the data is linearly separable.

Consider the inner product of \mathbf{w}_t and \mathbf{w}_f

$$\begin{aligned}\mathbf{w}_f^T \mathbf{w}_{t+1} &= \mathbf{w}_f^T (\mathbf{w}_t + \frac{1}{10} y_{n(t)} \mathbf{w}_t) \\ &\geq \mathbf{w}_f^T \mathbf{w}_t + \frac{1}{10} \min_n (y_n \mathbf{w}_f^T \mathbf{x}_n)\end{aligned}$$

Next, we consider the change of the length of \mathbf{w}_t .

$$\begin{aligned}\|\mathbf{w}_{t+1}\|^2 &= \left\| \mathbf{w}_t + \frac{1}{10} y_{n(t)} \mathbf{x}_{n(t)} \right\|^2 \\ &= \|\mathbf{w}_t\|^2 + \frac{1}{5} y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)} + \frac{1}{100} \|\mathbf{x}_{n(t)}\|^2 \leq \|\mathbf{w}_t\|^2 + \frac{1}{100} \max_n (\|\mathbf{x}_n\|^2)\end{aligned}$$

Now we consider the cosine value of the angle between \mathbf{w}_T and \mathbf{w}_f after T updates, assuming that the process starts with $\mathbf{w}_0 = \mathbf{0}$. Let $\rho = \frac{\min_n (y_n \mathbf{w}_f^T \mathbf{x}_n)}{10 \|\mathbf{w}_f\|}$ and $R^2 = \max_n (\|\mathbf{x}_n\|^2)$

$$\begin{aligned}& \frac{\mathbf{w}_f^T \mathbf{w}_T}{\|\mathbf{w}_f\| \|\mathbf{w}_T\|} \\ & \geq \frac{T\rho}{\|\mathbf{w}_T\|} \geq \frac{T\rho}{R\sqrt{T}} = \frac{\sqrt{T}\rho}{R}\end{aligned}$$

Since the cosine value must be no greater than 1, we have

$$\begin{aligned}\frac{\sqrt{T}\rho}{R} &\leq 1 \\ \Rightarrow T &\leq \left(\frac{R}{\rho}\right)^2\end{aligned}$$

We see that T is upper-bounded, so this process halts with a perfect hyperplane.

2.2

Now we consider the same process, only that we keep using the same example for update (that is, $n(t+1) = n(t)$ until the criteria for update is not met).

We now calculate the number of updates that is made with an example

$$\begin{aligned}& y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)} + \frac{n}{10} \|\mathbf{x}_{n(t)}\|^2 > 0 \\ \Rightarrow n &> -\frac{10 y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)}}{\|\mathbf{x}_{n(t)}\|^2} \\ \Rightarrow n &= \left\lceil -\frac{10 y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)}}{\|\mathbf{x}_{n(t)}\|^2} + 1 \right\rceil\end{aligned}$$

Hence after all the updates with one example $\mathbf{x}_{n(t)}$ is done, the new \mathbf{w} is

$$\mathbf{w} = \mathbf{w}_t + \frac{1}{10} y_{n(t)} \mathbf{x}_{n(t)} \left\lceil -\frac{10 y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)}}{\|\mathbf{x}_{n(t)}\|^2} + 1 \right\rceil$$

This is identical to the way \mathbf{w} is updated in the PLA invariant described in this problem.

2.3

Now we see that the PLA variant described in this problem is equivalent to the one we proved to halt to a perfect hyperplane in the previous part, only that we choose the examples with which we update \mathbf{w} in a special manner. From the proof presented in the previous part we see that how we choose the examples used for updates should not affect the fact that the process halts to a perfect hyperplane. Hence we've proven that this PLA process halts to a perfect hyperplane if the data is linearly separable.