ML homework 1: question 5

Q. What is a possible application of active learning?

The answer generated by chatGPT:

In medical imaging diagnosis, active learning is used to improve the labeling of medical images (e.g., MRI or CT scans) for diagnosing diseases like cancer or identifying abnormalities.

My opinion is:

I agree with ChatGPT's explanation because, firstly, active learning stands out from other protocols due to its unique ability to query only when the algorithm is "confused" or uncertain about a particular instance. This characteristic is especially valuable in applications like medical imaging, where identifying abnormalities (like in CT scans) is critical.

The primary advantage of this protocol is that, it significantly reduces the labeling burden. In many medical applications, labeling data requires the expertise of specialists such as doctors. Hiring professionals with such expertise to label large amounts of data can be both time-consuming and expensive, making it impractical for large datasets. By using active learning, only the most uncertain cases (such as CT scans that are difficult to distinguish from normal ones) need to be verified by the experts.

Secondly, active learning operates in a sequential manner. After the model identifies and queries the most uncertain case, an expert provides the label, and the model is immediately updated. This allows the algorithm to refine its understanding, so similar cases are less likely to be queried again. In contrast to batch learning, which updates the model with a fixed set of labeled data all at once, this sequential approach ensures that the expert is not asked to label similar cases repeatedly, therefore simplifying the process.

ML homework 1: question 6

Q. Can learning used to predict earthquakes?

The answer generated by chatGPT:

Machine learning can be used to predict earthquakes by analyzing seismic data to identify patterns that may precede earthquake events.

My opinion is:

My answer is yes, but only partially. Recent research has shown that by analyzing the acoustic signals generated in laboratory experiments, machine learning can predict the time remaining until an "earthquake" occurs with high accuracy. Other works usually aimed to focus on predicting specific features of an earthquake, like predicting the magnitude and depth.

While we can gain some insights into earthquakes, it seemed that there is currently no method to generalize these research findings to real-world scenarios. Moreover, there is likely no model capable of accurately predicting when and where an earthquake will happen, and how strong it will be. I think that the challenges are due to reasons like:

1. There are various phenomena that may be signals that represent the occurence of an earthquake, while we can indicate that an earthquake is going to happen directly by some of the phenomena like the p-wave, some of the other phenomena may have ambiguity in whether it can infer an earthquake or not, like a research I saw stated that the sound data they used are initially considered noise.

2. Machine learning models require sufficiently large datasets, if the number of data is too small, then there might be high generalization error, therefore there is difficulty obtaining enough usable data, even though there might be some data from the government, we still have to get data from different regions with varying characteristics in order to ensure that the dataset is comprehensive.

Thus, I would not say that machine learning is entirely ineffective in predicting earthquakes, but there are still many challenges to overcome. As we can see, people are dedicated to this field, so it is not impossible, and perhaps one day, machine learning will have practical applications in earthquake prediction.

7. Let $\vec{x_n}^{orig} = [x_{n1}\ x_{n2}\ \cdots\ x_{nd}] \in \mathbb{R}^d$ for $n \in \{1, \dots, N\}$

then $\vec{x_n} = [1\ x_{n1}\ x_{n2}\ \cdots\ x_{nd}] \in \mathbb{R}^{d+1}$ for $n \in [1, \dots, N]$

and $\vec{x_n}' = [2\ x_{n1}\ x_{n2}\ \dots\ x_{nd}] \in \mathbb{R}^{d+1}$ for $n \in [1, \dots, N]$

Suppose for the the $\vec{x_n}$s that are formed conventionally,

the first p-1 inputs are predicted correctly, then:

$$\text{sign}(\vec{w_0}^T \vec{x_i}) = y_i \quad \text{for } i = 0 \sim p\text{-}1$$

If we assume $\text{sign}(0) = -1$ for simplicity, and $\vec{w_0} = \vec{0}$, then this means that.

$$\text{sign}(\vec{0}^T \vec{x_i}) = \text{sign}(0) = -1 = y_i \quad \text{for } i = 1 \sim p\text{-}1$$

If we further assume $\vec{x_p}$ receives incorrect prediction, then

$$\text{sign}(\vec{0}^T \vec{x_p}) \neq y_p$$

$$\Rightarrow \text{sign}(0) = -1 \neq y_p \quad \therefore y_p = 1$$

Thus we update $\vec{w_1}$ as:

$$\vec{w_1} = \vec{w_0} + y_p \vec{x_p} = \vec{0} + 1 \cdot \begin{bmatrix} 1 \\ x_{p1} \\ x_{p2} \\ \vdots \\ x_{pd} \end{bmatrix} = \begin{bmatrix} 1 \\ x_{p1} \\ x_{p2} \\ \vdots \\ x_{pd} \end{bmatrix}$$

Since $\vec{w_0}' = \vec{0}$, and y_i remain unchanged no matter we use $\vec{x_i}$ or $\vec{x_i}'$, so:

$$\text{sign}(\vec{w_0}'^T \vec{x_i}') = \text{sign}(\vec{0}^T \vec{x_i}') = -1 = y_i \quad \text{for } i = 1, \dots, p-1$$

We next want to know if $\vec{x_p}'$ will get correct prediction or not:

$$\text{sign}(\vec{w_0}'^T \vec{x_p}') = \text{sign}(\vec{0}^T \vec{x_p}') = -1 \neq 1 = y_p$$

Thus $\vec{x_p}'$ is also mispredicted, and we update the weight vector as:

$$\vec{w_1}' = \vec{w_0}' + y_p \vec{x_p}' = \vec{0} + 1 \cdot \begin{bmatrix} 2 \\ x_{p1} \\ x_{p2} \\ \vdots \\ x_{pd} \end{bmatrix} = \begin{bmatrix} 2 \\ x_{p1} \\ x_{p2} \\ \vdots \\ x_{pd} \end{bmatrix}$$

Consider the next input $\vec{x_{p+1}}$ and $\vec{x_{p+1}}'$:

$$\boxed{\text{sign}(\vec{w_1}^T \vec{x_{p+1}})} = \text{sign}\left([1 \ x_{p1} \ x_{p2} \cdots x_{pd}] \begin{bmatrix} 1 \\ x_{p+1,1} \\ x_{p+1,2} \\ \vdots \\ x_{p+1,d} \end{bmatrix} \right)$$

$$= \underline{\text{sign}\left(1 + \sum_{i=1}^{d} x_{pi} x_{p+1,i}\right)}$$

$$\boxed{\text{sign}(\vec{w_1}'^T \vec{x_{p+1}}')} = \text{sign}\left([2 \ x_{p1} \ x_{p2} \cdots x_{pd}] \begin{bmatrix} 2 \\ x_{p+1,1} \\ x_{p+1,2} \\ \vdots \\ x_{p+1,d} \end{bmatrix} \right)$$

$$= \underline{\text{sign}\left(4 + \sum_{i=1}^{d} x_{pi} x_{p+1,i}\right)}$$

Therefore, consider the situation that if $\sum_{i=1}^{d} x_{pi} x_{p+1,i} = -3$

then:

$$\text{sign}(\vec{w_1}^T \vec{x}_{p+1}) = \text{sign}(1+(-3)) = \text{sign}(-2) = -1$$
$$\text{sign}(\vec{w_1}'^T \vec{x}_{p+1}') = \text{sign}(4+(-3)) = \text{sign}(1) = 1$$

$\Rightarrow$ This will cause $\vec{w_1}$ to update if $y_{p+1} = 1$, $\vec{w_1}'$ to update if $y_{p+1} = -1$.

If the case is $y_{p+1} = -1$, and there are only $p+1$ point in the dataset,

(i.e. $\{\vec{x}_n, y_n\}_{n=1}^{p+1}$ / $\{\vec{x}_n', y_n\}_{n=1}^{p+1}$)

then $\vec{w}_{PLA} = \vec{w_1}$

and we continue to update $\vec{w_1}'$ and check for $\vec{x}_1'$:

$$\vec{w}_2' = \vec{w_1}' + y_{p+1} \cdot \vec{x}_{p+1}' = \begin{bmatrix} 2 \\ x_{p1} \\ x_{p2} \\ \vdots \\ x_{pd} \end{bmatrix} + (-1) \cdot \begin{bmatrix} 2 \\ x_{p+1,1} \\ x_{p+1,2} \\ \vdots \\ x_{p+1,d} \end{bmatrix} = \begin{bmatrix} 0 \\ x_{p1} - x_{p+1,1} \\ x_{p2} - x_{p+1,2} \\ \vdots \\ x_{pd} - x_{p+1,d} \end{bmatrix}$$

$$\text{sign}(\vec{w}_2'^T \vec{x}_1') = \text{sign}\left(\begin{bmatrix} 0 & x_{p1} - x_{p+1,1} & \cdots & x_{pd} - x_{p+1,d} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1d} \end{bmatrix} \right)$$

$$= \text{sign}\left(\sum_{i=1}^{d} (x_{pi} - x_{p+1,i}) x_{1i} \right)$$

If $\sum_{i=1}^{d}(x_{pi} - x_{p+1,i})x_{1i} \le 0$, and since $y_1 = -1$ as we assumed earlier, we have correct prediction using $\vec{w_2}'$.

We can imagine that we construct the inputs to make :

$$\text{sign}(\vec{w_2}'^T \vec{x_i}) = y_i \quad \text{for } i = 1, \dots, p+1$$

then $$\vec{w_{PLA}}' = \vec{w_2}' = \begin{bmatrix} 0 \\ x_{p1} - x_{p+1,1} \\ \vdots \\ x_{pd} - x_{p+1,d} \end{bmatrix}$$

or we can see it is trivial that each time we update $\vec{w_i}'$, we add / substract 2 to w_{i0}, thus

$$\vec{w_{PLA}}' = \begin{bmatrix} 2k \\ \vdots \\ \vdots \end{bmatrix} \quad \text{for } k \in \mathbb{Z}$$

and $\vec{w_{PLA}} = \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}$, so $\vec{w_{PLA}}' \neq \vec{w_{PLA}}$. $\square$

8.

Let $\vec{x}_n^{orig} = [x_{n1}\ x_{n2}\ \cdots\ x_{nd}] \in \mathbb{R}^d$

$\Rightarrow \begin{cases} \vec{x}_n = [1\ x_{n1}\ x_{n2}\ \cdots\ x_{nd}] \in \mathbb{R}^{d+1} \\ \vec{x}_n' = [3\ \ 3x_{n1}\ 3x_{n2}\ \cdots\ 3x_{nd}] \in \mathbb{R}^{d+1} \end{cases}$

For arbitrary $(\vec{x}_n, y_n) \in \{(\vec{x}_n, y_n)\}_{n=1}^{N}$, and assume $\text{sign}(0) = 1$

if $\text{sign}(\vec{w}_0^T \vec{x}_p) \neq y_p$, which means $\vec{x}_p$ is the first mispredicted input

then $y_p = 1$

and

$$\vec{w}_1 = \vec{w}_0 + y_n \vec{x}_p = \vec{x}_p$$

If we further assume the second misprediction occurs at $(\vec{x}_q, y_q)$, then

$\text{sign}(\vec{w}_1^T \vec{x}_q) \neq y_q$

$\Rightarrow \text{sign}(\vec{x}_p^T \vec{x}_q) \neq y_q$

$\Rightarrow \text{sign}\left([1\ x_{p1}\ x_{p2}\ \cdots\ x_{pd}] \begin{bmatrix} 1 \\ x_{q1} \\ x_{q2} \\ \vdots \\ x_{qd} \end{bmatrix}\right) \neq y_q$

$\Rightarrow \text{sign}\left(1 + \sum_{i=1}^{d} x_{pi} x_{qi}\right) \neq y_q$

$\because$ $\vec{x}_0 \sim \vec{x}_{p-1}$ is predicted correctly by $\vec{w}_0$

$\therefore$ $\text{sign}(\vec{w}_0^T \vec{x}_i) = y_i$ for $i = 1 \sim p-1$

$\Rightarrow$ $\text{sign}(\vec{w}_0^T \vec{x}_i) = y_i$

$\Rightarrow$ $\text{sign}\left([0\ 0\ \cdots\ 0] \begin{bmatrix} 1 \\ x_{i1} \\ x_{i2} \\ \vdots \\ x_{id} \end{bmatrix} \right) = y_i$

$\Rightarrow$ $\text{sign}(0) = -1 = y_i$ for $i = 1 \sim p-1$

$\because$ $\text{sign}(\vec{w}_0^T \vec{x}_p) \neq y_p$ as we assumed $\therefore$ $y_p = 1$

Thus, $\vec{w}_1 = \vec{w}_0 + y_p \vec{x}_p = \vec{0} + 1 \cdot \begin{bmatrix} 1 \\ x_{p1} \\ x_{p2} \\ \vdots \\ x_{pd} \end{bmatrix} = \begin{bmatrix} 1 \\ x_{p1} \\ x_{p2} \\ \vdots \\ x_{pd} \end{bmatrix}$

Consider the same situation but replace all $\vec{x}_n$ to $\vec{x}_n'$:

For $\vec{x}_i'$ $i = 1 \sim p-1$, since $\text{sign}(\vec{w}_0'^T \vec{x}_i') = -1$ and y_i remain unchanged

$\therefore$ $\vec{x}_1' \sim \vec{x}_{p-1}'$ is predicted correctly

For $\vec{x}_p'$, we have also have $\text{sign}(\vec{w}_0'^T \vec{x}_p') = -1 \neq y_p = 1$

Therefore,

$$\vec{w}_1' = \vec{w}_0' + y_p \vec{x}_p' = \vec{0} + 1 \cdot \begin{bmatrix} 3 \\ 3x_{p1} \\ 3x_{p2} \\ \vdots \\ 3x_{pd} \end{bmatrix} = \begin{bmatrix} 3 \\ 3x_{p1} \\ 3x_{p2} \\ \vdots \\ 3x_{pd} \end{bmatrix}$$

$\because$ $\vec{x}_{p+1} \sim \vec{x}_{q-1}$ is predicted correctly

$\therefore$ $\text{sign}(\vec{x}_p^T \vec{x}_i) = y_i$ for $i = p+1 \sim q-1$

We want to know if $\vec{x}_{p+1}' \sim \vec{x}_{q-1}'$ will be predicted correctly or not

as $\vec{w}_1' = [3 \;\; 3x_{p1} \cdots 3x_{pd}]^T$

$$\Rightarrow \text{sign}(\vec{w}_1'^T \vec{x}_{p+1}') = \text{sign}\left([3 \;\; 3x_{p1} \cdots 3x_{pd}] \begin{bmatrix} 3 \\ 3x_{p+1,1} \\ 3x_{p+1,2} \\ \vdots \\ 3x_{p+1,d} \end{bmatrix} \right)$$

$$= \text{sign}(9(1 + x_{p1}x_{p+1,1} + \cdots + x_{pd}x_{p+1,d}))$$

$$= \text{sign}(1 + x_{p1}x_{p+1,1} + \cdots + x_{pd}x_{p+1,d})$$

$$= \text{sign}(\vec{x}_p^T \vec{x}_{p+1})$$

$$= \text{sign}(\vec{w}_1^T \vec{x}_{p+1})$$

similarly, we know that:

$$\text{sign}(\vec{w}_1'^T \vec{x}_i') = \text{sign}(\vec{w}_1^T \vec{x}_i) \text{ for } i = p+1 \sim q-1$$

Thus, $\vec{x}_i'$ is predicted correctly by $\vec{w}_1'$ for $i = p+1 \sim q-1$

Next we want to check if $\text{sign}(\vec{w}_1'^T \vec{x}_q') = y_q$ or not:

$$\text{sign}(\vec{w}_1'^T \vec{x}_q) = \text{sign}\left([3 \;\; 3x_{p1} \cdots 3x_{pd}] \begin{bmatrix} 3 \\ 3x_{q1} \\ 3x_{q2} \\ \vdots \\ 3x_{qd} \end{bmatrix} \right)$$

$$= \text{sign}(9(1 + x_{p1}x_{q1} + \cdots + x_{pd}x_{qd}))$$

$$= \text{sign}\left(9\left(1 + \sum_{i=1}^{d} x_{pi}x_{qi}\right)\right)$$

$$= \text{sign}\left(1 + \sum_{i=1}^{d} x_{pi}x_{qi}\right)$$

$$= \operatorname{sign}(\vec{x_p}^T \vec{x_q})$$

$$= \operatorname{sign}(\vec{w_1}^T \vec{x_q})$$

Thus, if $\operatorname{sign}(\vec{w_1}^T \vec{x_q}) \neq y_q$, then $\operatorname{sign}(\vec{w_1}'^T \vec{x_q}') \neq y_q$

Therefore, we can know that $\vec{w}_{PLA} = \vec{w}'_{PLA}$.

9. Let article : $\vec{x}_t = [1 \ \ x_{t1} \ \ x_{t2} \cdots x_{td}]^T \quad t \in \{1, \dots, \overset{\#\text{articles}}{\boxed{N}}\}$

dictionary : D , where $|D| = d \geq m$

$$D = [\underbrace{w_1 \cdots w_{d_+}}_{\text{more hatred like words}} \underbrace{w_{d_+ +1} \cdots \overset{w_d}{\overset{\parallel}{w_{d_+ + d_-}}}}_{\text{less hatred like words}}]^T$$

$\forall x_{ti} \ \ i \in \{1, \dots, d\} \quad x_{ti} = 1$ if $w_i \in$ article t

$$\begin{cases} z_+(\vec{x}_t) = \sum_{i=1}^{d_+} x_{ti} \\ z_-(\vec{x}_t) = \sum_{i=d_+ +1}^{d} x_{ti} \end{cases}$$

The mistake bound of PLA $= \dfrac{R^2}{\rho^2}$, where $\begin{cases} R^2 = \max\limits_n \|\vec{x}_n\|^2 \\ \rho = \min\limits_n y_n \dfrac{\vec{w}_f^T}{\|\vec{w}_f\|} \vec{x}_n \end{cases}$

$\because \|\vec{w}_f\| = \sqrt{\vec{w}_f^T \vec{w}_f}$

$\therefore \|\vec{w}_f\|^2 = \vec{w}_f^T \vec{w}_f$

$$\Rightarrow \rho^2 = \min_n \frac{(\vec{w}_f^T \vec{x}_n)^2}{\vec{w}_f^T \vec{w}_f}$$

Since we count number of hatred words to determine if an article is hatred or not, and the bias is -3.5, $\vec{w_f}$ is:

$$\vec{w_f} = [-3.5\ \overbrace{\underbrace{+1\ +1\ \cdots\ +1}_{d+}\ \underbrace{-1\ -1\ \cdots\ -1}_{d-}}^{d}]^T \in \mathbb{R}^{d+1}$$

$$\left(\vec{x} = [x_0\ x_1 \cdots x_d]^T = [1\ x_1 \cdots \boxed{x_d}]^T \right)$$

$\hookrightarrow$ 如果word i 在 article $\vec{x}$ 中 $\Rightarrow x_i = 1$ for $i = 1, \ldots, d$

$$\therefore \|\vec{w_f}\| = \sqrt{\vec{w_f}^T \vec{w_f}} = \sqrt{12.25 + 1^2 \times d+ + (-1)^2 \times d-} = \sqrt{12.25 + d}$$

$$\Rightarrow \rho^2 = \min_n \frac{(\vec{w_f}^T \vec{x_n})^2}{\vec{w_f}^T \vec{w_f}} = \min_n \frac{(\vec{w_f}^T \vec{x_n})^2}{12.25 + d} = \frac{0.25}{12.25 + d}$$

suppose $d+ > d-$
则min发生在 $\vec{x} =$

$$[1\ \underbrace{11 \cdots 1}_{\min(d+,d-)}\ 111 \cdots 0\ \underbrace{11 \cdots 1}_{\min(d+,d-)}]^T$$

$\min(d+,d-) = d-$, $\min(d+,d-) = d-$

$$\therefore \min_n (\vec{w_f}^T \vec{x_n})^2 = \min_n (-3.5 + d- + 3 + 0 - d-)^2 = \min_n (-0.5)^2 = 0.25$$

$$R^2 = \max_n \|\vec{x_n}\|^2 = \max_n \vec{x_n}^T \vec{x_n}$$

取 $\vec{x_n} = [\underset{x_0}{1}\ \underbrace{11 \cdots 1}_{m\text{ 個}}\ 0 \cdots 0]^T$

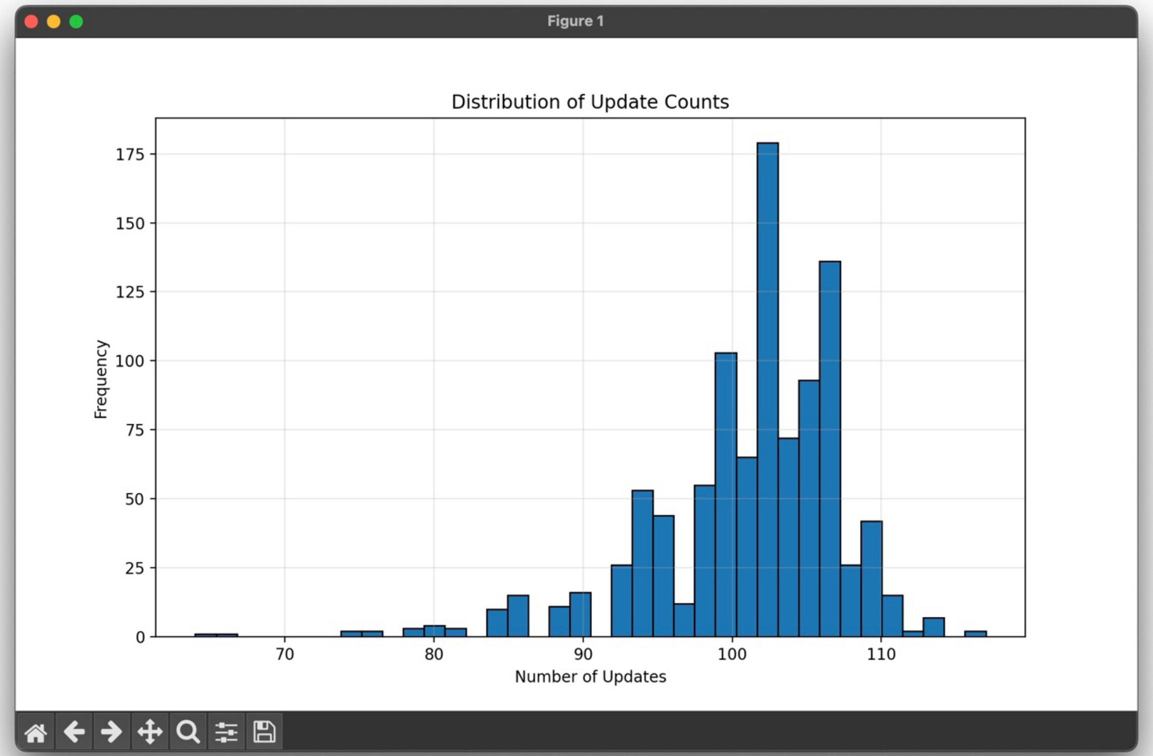
$$\therefore R^2 = m + 1$$

Therefore, the upperbound of mistake number $= \dfrac{m+1}{\dfrac{0.25}{12.25+d}}$ $\square$

ML homework 1: Question 10

Part 1: Interpretation

The resulting histogram generated by my code is as below:



From the histogram, we can see that the peak is about 104, and most of the number of updates are around 95 to 110. As what we learned in class, the upper bound T is proportional to R^2, if we consider the high dimensionality of the inputs, we might expect T to be a considerably large value, which does not match the result of the distribution. I think the reason would be the sparseness of the inputs, which makes R^2 much smaller than a 47206 dimension input should probably have.

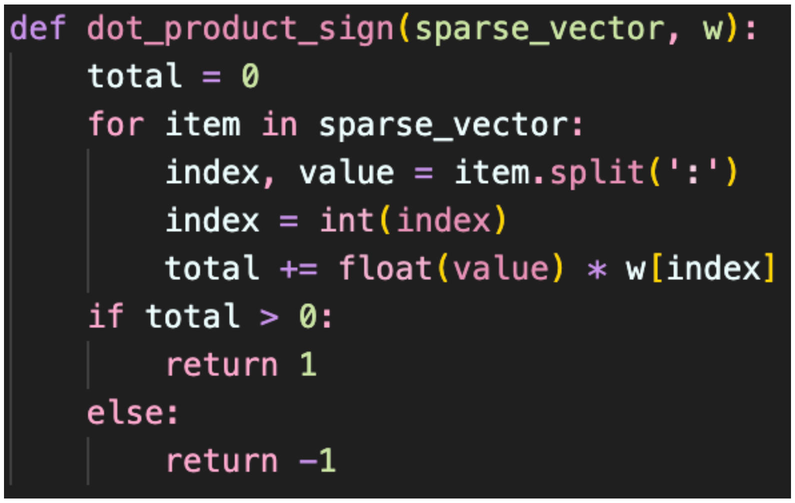
Another interpretation is that, since the input vector with the 1 added as the zero-th term is of dimension 47206, we knew that the weight vector is of same dimension, which means we have high flexibility to modify our perceptron. In such a high dimensional space, finding a hyperplane to separate the data points seemed to be feasible, and this reflects in our histogram, where most number of updates are relatively low, indicating that we can achieve 99% of accuracy in around one hundred updates.

However, there seemed to be some extreme cases, like the experiments that have update number higher than 110. This is probably due to the random choosing process that we use allows replacement, also, the dataset is small (200) compared to the dimension (47206), therefore, if in some situation we chose the points near the decision boundary many times, it may be difficult for the PLA to get 1000 consecutive correct predictions.

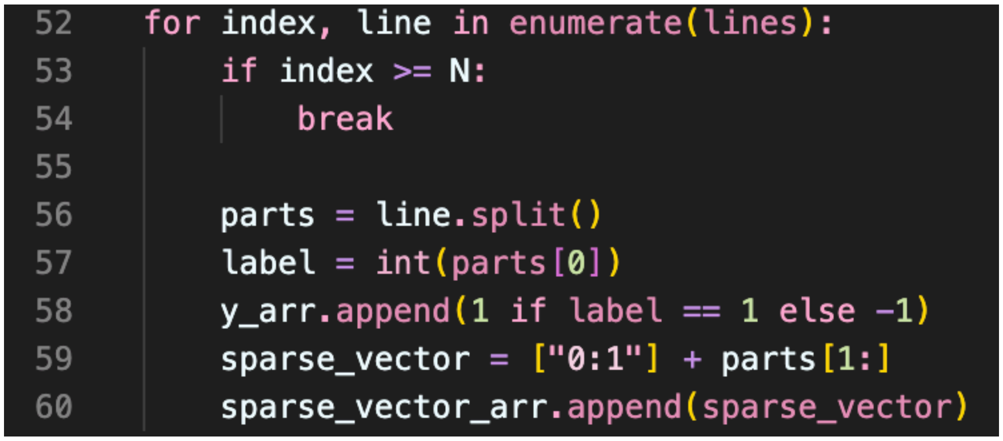
For the cases that the number of updates are smaller than 90, it might because the similar reason as the extreme large cases we stated above. Since the dataset is small, it is more possible that we're "fortunate" enough to choose the points with large distance repeatedly, so that PLA would converge quickly.

Part 2: Code snippet

The following pictures are some of my code snippets. Since I wrote lots of comments, providing only the first page would show quite little code that provides real usage:

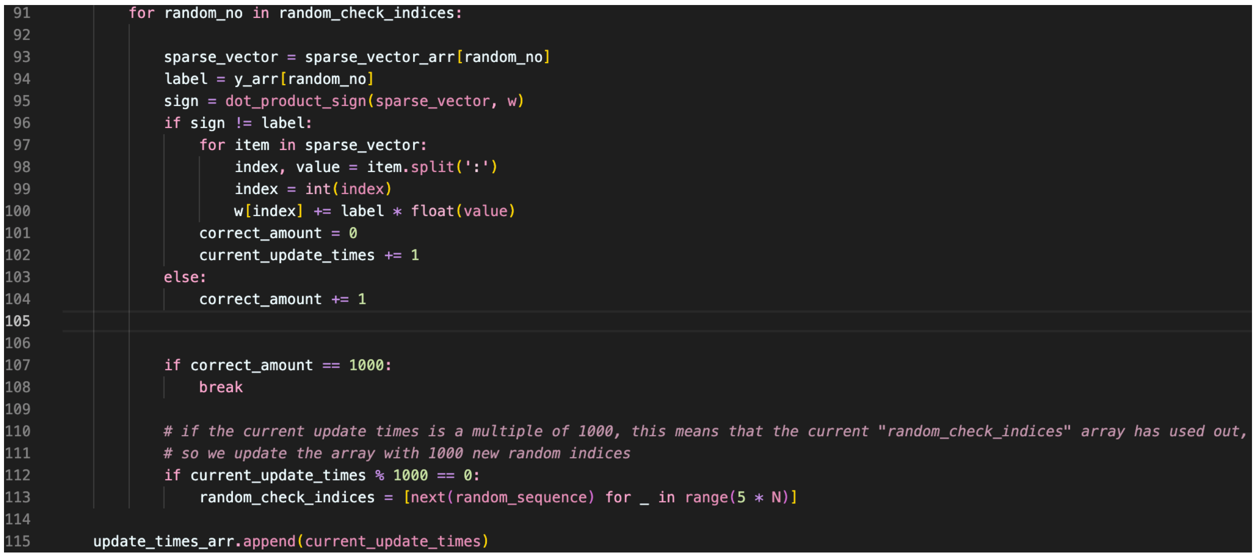


```python
def dot_product_sign(sparse_vector, w):
    total = 0
    for item in sparse_vector:
        index, value = item.split(':')
        index = int(index)
        total += float(value) * w[index]
    if total > 0:
        return 1
    else:
        return -1
```



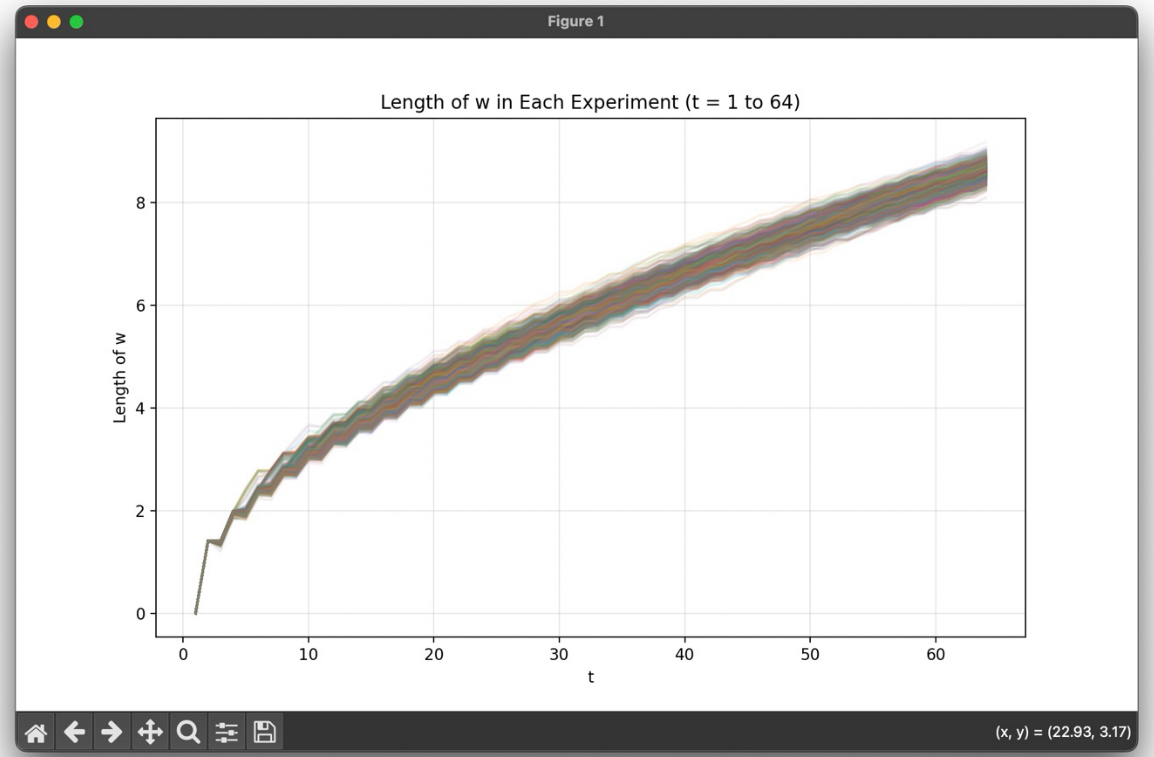
```python
    for index, line in enumerate(lines):
        if index >= N:
            break

        parts = line.split()
        label = int(parts[0])
        y_arr.append(1 if label == 1 else -1)
        sparse_vector = ["0:1"] + parts[1:]
        sparse_vector_arr.append(sparse_vector)
```

The following part is contained in two loops, where the outer one does the experiment 1000 times, and the inner one loops while the correct amount is smaller than 1000:



```python
        for random_no in random_check_indices:

            sparse_vector = sparse_vector_arr[random_no]
            label = y_arr[random_no]
            sign = dot_product_sign(sparse_vector, w)
            if sign != label:
                for item in sparse_vector:
                    index, value = item.split(':')
                    index = int(index)
                    w[index] += label * float(value)
                correct_amount = 0
                current_update_times += 1
            else:
                correct_amount += 1


            if correct_amount == 1000:
                break

            # if the current update times is a multiple of 1000, this means that the current "random_check_indices" array has used out,
            # so we update the array with 1000 new random indices
            if current_update_times % 1000 == 0:
                random_check_indices = [next(random_sequence) for _ in range(5 * N)]

    update_times_arr.append(current_update_times)
```

ML homework 1: question 11



Findings:

The minimum number of updates $T_{\min}$ (or min_T in my code) I got is 64.

From this graph, we can see that the growth of the length of the weight vector is slow, and the length until the 64-th update did not exceed 10. This might reflect what we learned in class: the square of the growing length in each update is bounded by R^2.

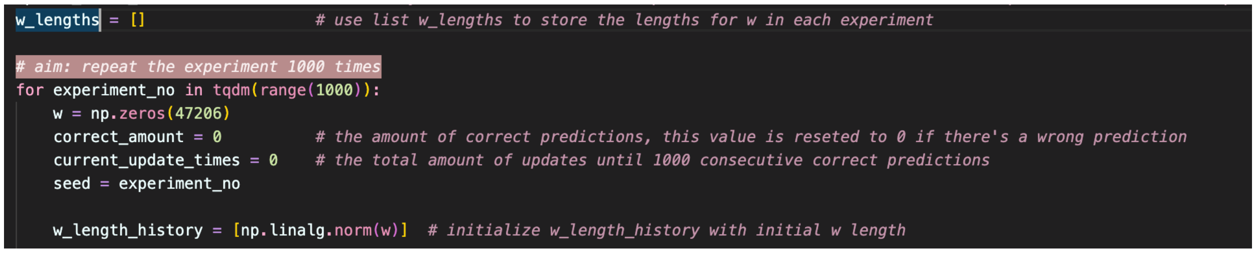
However, we can still see that compare to the later updates, the length grows faster in the first 10 updates. This may due to the fact that the initial value of the weight vector is a zero vector, so that it is far from the ultimate w_{PLA}, hence making bigger changes in the first few updates. The growth became more stable after some updates, it is probably because after the first few updates, we have constructed a hyperplane that can classify most of the examples correctly, so the amplitude of the adjustments became small.

Moreover, we can see that the variation of the 1000 experiments is larger in the first few updates, but as the number of update (t) increases, the variations became small, indicating that we get similar weight lengths after a certain number of updates. I think that the reason is because we got 99% accuracy after 1000 consecutive correct predictions, and the dataset we use is the same, so that if the data points are linear separable, then there exists a hyperplane that can correctly classify all the points, even if there is some noise, we still can get a hyperplane that can separate most of the points. Therefore, as the high accuracy we got in each experiment, the weight vector we got will represent similar hyperplanes, hence resulting a similar value of the weight vector length.

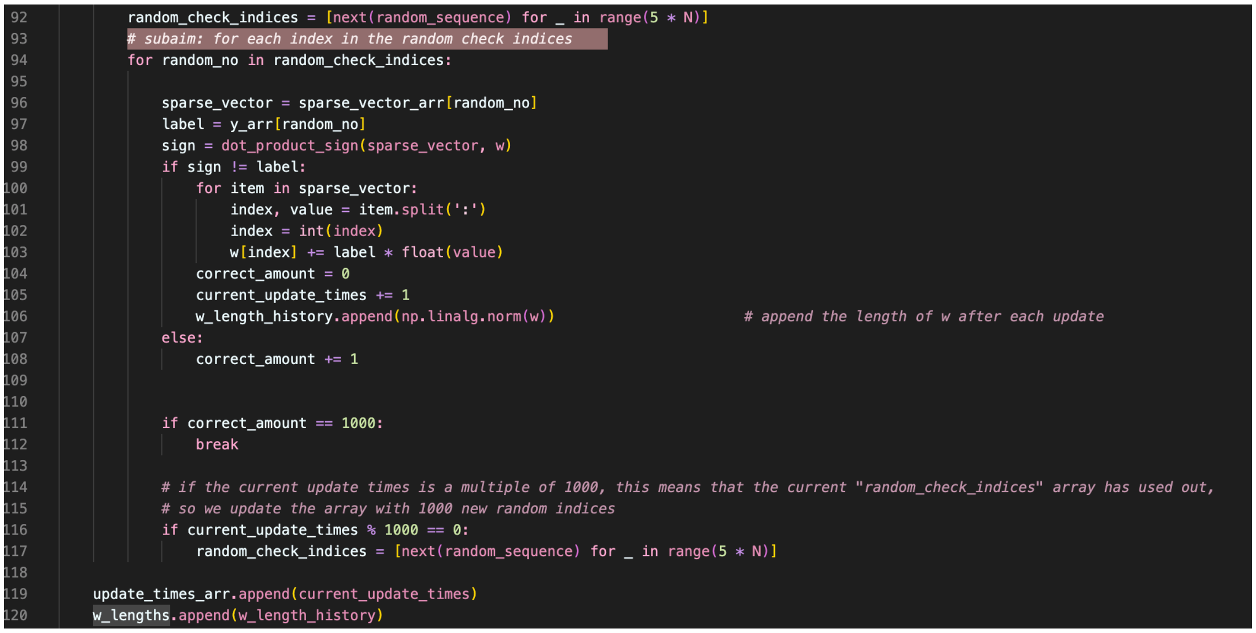
Part of code:

Most of the code in this question is similar to question 10, I just add some modifications to record the history length of $\overrightarrow{w}$, take the minimum value from the resulting value of Ts, and change the figure plotting part.

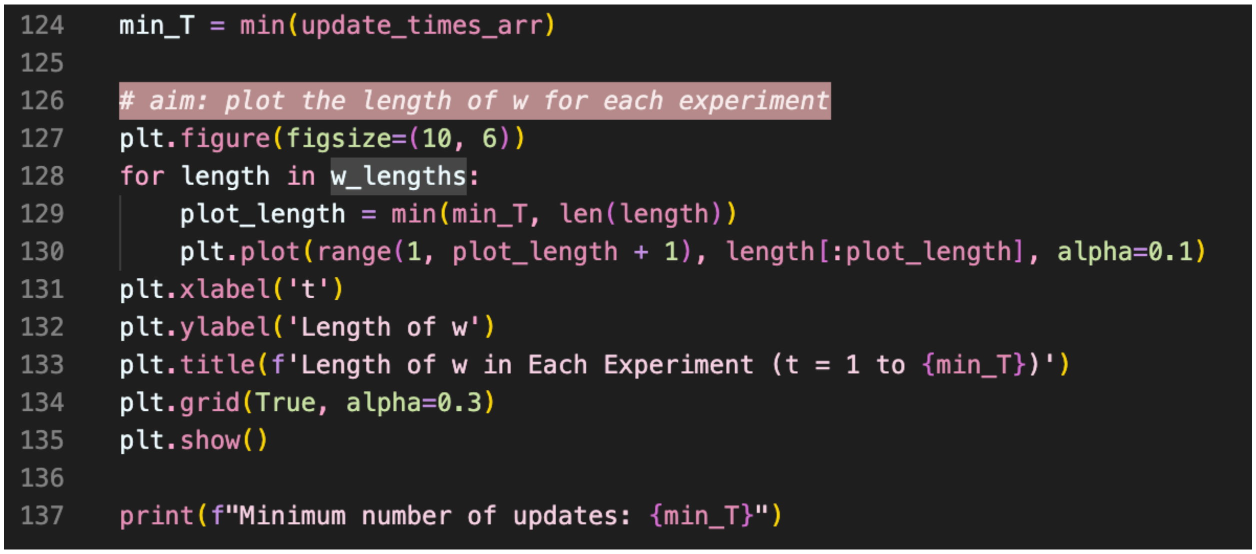
Due to the same problem as question 10, there's too many comments in my code, so I would provide the snapshots of the parts that include the modifications as below:



```
w_lengths = []                  # use list w_lengths to store the lengths for w in each experiment

# aim: repeat the experiment 1000 times
for experiment_no in tqdm(range(1000)):
    w = np.zeros(47206)
    correct_amount = 0          # the amount of correct predictions, this value is reseted to 0 if there's a wrong prediction
    current_update_times = 0    # the total amount of updates until 1000 consecutive correct predictions
    seed = experiment_no

    w_length_history = [np.linalg.norm(w)]  # initialize w_length_history with initial w length
```



```python
        random_check_indices = [next(random_sequence) for _ in range(5 * N)]
        # subaim: for each index in the random check indices
        for random_no in random_check_indices:

            sparse_vector = sparse_vector_arr[random_no]
            label = y_arr[random_no]
            sign = dot_product_sign(sparse_vector, w)
            if sign != label:
                for item in sparse_vector:
                    index, value = item.split(':')
                    index = int(index)
                    w[index] += label * float(value)
                correct_amount = 0
                current_update_times += 1
                w_length_history.append(np.linalg.norm(w))                      # append the length of w after each update
            else:
                correct_amount += 1


            if correct_amount == 1000:
                break

            # if the current update times is a multiple of 1000, this means that the current "random_check_indices" array has used out,
            # so we update the array with 1000 new random indices
            if current_update_times % 1000 == 0:
                random_check_indices = [next(random_sequence) for _ in range(5 * N)]

    update_times_arr.append(current_update_times)
    w_lengths.append(w_length_history)
```



```python
min_T = min(update_times_arr)

# aim: plot the length of w for each experiment
plt.figure(figsize=(10, 6))
for length in w_lengths:
    plot_length = min(min_T, len(length))
    plt.plot(range(1, plot_length + 1), length[:plot_length], alpha=0.1)
plt.xlabel('t')
plt.ylabel('Length of w')
plt.title(f'Length of w in Each Experiment (t = 1 to {min_T})')
plt.grid(True, alpha=0.3)
plt.show()

print(f"Minimum number of updates: {min_T}")
```