Homework # 1

姓名	學號
操之晴	R13922A04

5. The answer ChatGPT gave me was “medical image analysis”, such as for cancer detection. It’s convincing because “active learning” is especially useful when labeled data is scarce or expensive to obtain. Imagine a situation where we need a few expert radiologists to label every medical image, it can be both time-consuming and costly.

   In this case, active learning can help by allowing the model to identify the most uncertain or difficult cases for labeling, reducing the number of images that need to be labeled while still maintaining high performance. This makes the process more efficient and improves accuracy while minimizing the effort required from experts.

   For example, the model can select images that are most likely to be misclassified or those that provide the most value in distinguishing between cancerous and healthy tissues.
   That’s why I agree with this answer. Medical image analysis is a possible application of active learning.

6. The answer ChatGPT gave me was “Yes”, and it listed several ways machine learning can be applied to earthquake prediction: Seismic Data Analysis, Early Warning Systems, Identifying Risk Zones and Aftershock Prediction.

   I partially agree with this answer. Since the problem was about “earthquakes prediction”, the focus should be on predicting earthquakes that haven’t happened yet. In that sense, only the first point “Seismic Data Analysis” is directly relevant.

   Machine learning models can analyze large amounts of historical seismic data to detect patterns that might signal an upcoming earthquake. This could help find precursor signals, such as specific types of seismic waves, that suggest a higher risk of earthquake. However, predicting the exact time and location of an earthquake is still a huge challenge. That’s why more attention is given to the other tasks listed above.

7. (convention 0: sign(0) = 0)
Suppose followings are two data points and its label:

data points x	$x_1 = (-1,2)$	$x_2 = (1,-1)$
Label y	+1	-1
$x_0 = 1$	$(1,-1,2)$	(1, 1,-1)
	$w_1 = w_0 + y_1x_1 = (1,-1,2)$	$w_2 = w_1 = (1,-1,2)$
$x_0' = 2$	$(2,-1,2)$	(2, 1,-1)
	$w_1 = w_0 + y_1x_1 = (2,-1,2)$	$w_2 = w_1 + y_2x_2 = (0,-2,3)$

From above, $w_{PLA} = (1,-1,-1)$ and $w'_{PLA} = (0,-2,-2)$. When we take $x_3 =$ (2,1), in $x_0 = 1$, $sign\big((1,-1,2)\cdot(1,2,1)\big) = 1$. In $x'_0 = 2$, $sign\big((0,-2,3)\cdot(2,2,1)\big) = -1$. They return in different binary classification output. Therefore, they are not equivalent.

8. When we take $x_n' = 3x_n$ $w_0 = 0$ convention 1: sign(0) = 1
$w_1 = w_0 + y_1x_1$ $w_1' = w_0 + y_1x_1' = y_1 3x_1 = 3w_1$

For next x_2, we have two possible cases:

Case 1: sign($w_1 \cdot x_2$) $\neq y_2$

$$w_2 = w_1 + y_2x_2$$

Since $sign(w_1' \cdot x_2) = sign(3w_1 \cdot x_2) = sign(w_1 \cdot x_2) \neq y_2$

$$w_2' = w'_1 + y_2x_2' = 3w_1 + y_2 3x_2 = 3w_2$$

Case 2: sign($w_1 \cdot x_2$) $= y_2$

$$w_2 = w_1$$

Since $sign(w_1' \cdot x_2) = sign(3w_1 \cdot x_2) = sign(w_1 \cdot x_2) = y_2$

$$w_2' = w'_1 = 3w_1 = 3w_2$$

We can see that no matter the next sample is misclassified or not, $w' = 3w$. Which leads to $w'_{PLA} = 3w_{PLA}$. Then for every possible sample x_k in $\mathbb{R}^d$,

$$sign(w'_{PLA} \cdot x'_k) = sign(3w_{PLA} \cdot 3x_k) = sign(9w_{PLA} \cdot x_k) = sign(w_{PLA} \cdot x_k)$$

Therefore, w_{PLA} and w'_{PLA} are equivalent.

9. $T \leq \frac{R^2}{\rho^2}$ z_+: $h$atred − like z_-: $l$ess hatred − like

$$f(x) = sign(z_+(x) - z_-(x) - 3.5)$$

$$R^2 = \max\left||x_n|\right|^2 = \sqrt{(m+1)}^2 = m + 1$$

$$\rho = \min y_n \frac{w_f^T}{\left||w_f|\right|} x_n$$

$$\rho^2 = \frac{1}{\sqrt{d + 3.5^2}^2} \min\left(y_n w_f^T x_n\right)^2 = \frac{1}{d + 12.25} \min\left((z_+(x) - z_-(x) - 3.5)\right)^2$$

When $z_+ = z_-$, we get minimum $\rho^2 = \frac{1}{d+12.25} 3.5^2 = \frac{12.25}{d+12.25} = \frac{49}{4d+49}$

Thus, the upper bound is $T \leq \frac{R^2}{\rho^2} = \frac{1}{49}(4d + 49)(m + 1)$

10.

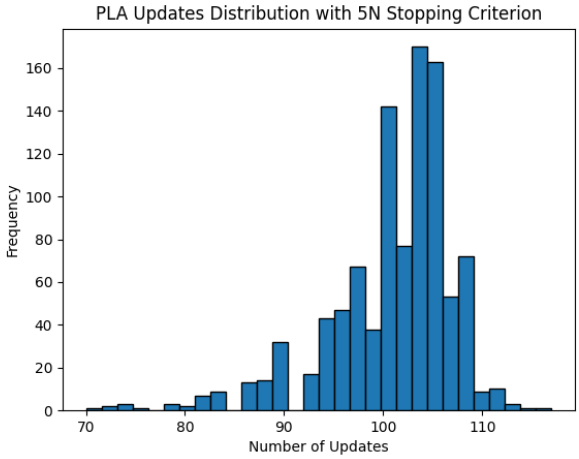

Figure 10a. updates histogram

```python
import numpy as np
import random
from sklearn.datasets import load_svmlight_file
import matplotlib.pyplot as plt

# parameters
experiment = 1000
sample = 200
#max_update = 5000
update_counts = []      # save every experiment update counts

# loading datas
def load_data(path, sample):
    x, y = load_svmlight_file(path)     # loading LIBSVM form
    x = x[:sample].toarray()        # to numpy array
    y = y[:sample]
    return x, y

x, y = load_data('rcv1_train.binary', sample)

# adding bias x0 = 1 to every xn
x =  np.hstack([np.ones((x.shape[0], 1)), x])

# PLA
def pla(x,y):
    w = np.zeros(x[0].shape)       # initialize w = 0
    updates = 0
    sample = x.shape[0]
    max_5N = 5 * sample
    no_error = 0

    while True:
        i = np.random.choice(sample)   # with replacement
        if np.sign(np.dot(w, x[i])) != y[i]:    # sign different
            w = w + y[i] * x[i]
            updates += 1
            no_error = 0
        else:
            no_error += 1
        if no_error >= max_5N:
```

Figure 10b. snapshot of my first page of code

a. Distribution Range

From figure 10a , we can see that the histogram has a positive skew, with most experiments needing around 100 to 110 updates. This suggests that it typically takes about that many updates for the PLA to converge and correctly classify all the samples. With this tight range, I found PLA is consistent in its performance on this dataset, usually requiring a similar number of updates.

b. Some Outliers

A few experiments needed more than 110 updates, likely because the algorithm ran into more difficult-to-classify samples, which required extra updates to resolve.

In summary, most experiments converge in around 100 updates, showing that this is the typical behavior of the PLA on this dataset. Although a few cases needed significantly more updates due to the randomness in sample selection, those were rare.

For problems 10 to 12, the full code has been made available on GitHub.
Feel free to check it out if required or interested!

11.

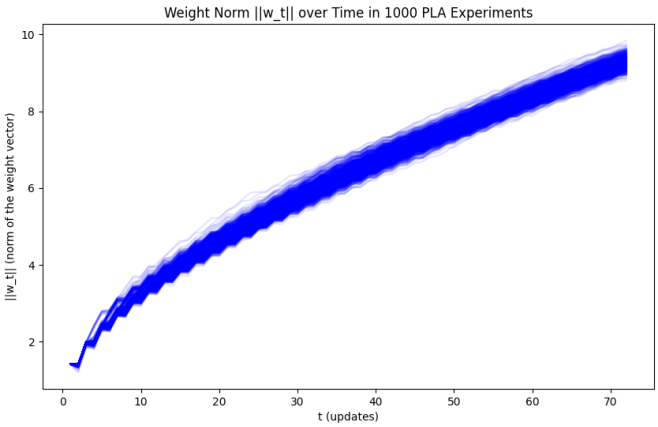

Figure 11a. $||w_t||$ vs. t

```python
import numpy as np
import random
from sklearn.datasets import load_svmlight_file
import matplotlib.pyplot as plt

# parameters
experiment = 1000
sample = 200
#max_update = 5000
update_counts = []      # save every experiment update counts
all_w_norm = []         # save every experiment w norms
min_update = np.inf     # save minimum updates


# loading datas
def load_data(path, sample):
    x, y = load_svmlight_file(path)     # loading LIBSVM form
    x = x[:sample].toarray()        # to numpy array
    y = y[:sample]
    return x, y

x, y = load_data('rcv1_train.binary', sample)

# adding bias x0 = 1 to every xn
x =  np.hstack([np.ones((x.shape[0], 1)), x])

# PLA
def pla(x,y):
    w = np.zeros(x[0].shape)       # initialize w = 0
    updates = 0
    sample = x.shape[0]
    max_5N = 5 * sample
    no_error = 0
    w_norm = []

    while True:
        i = np.random.choice(sample)   # with replacement
        if np.sign(np.dot(w, x[i])) != y[i]:    # sign different
            w = w + y[i] * x[i]
            updates += 1
```

Figure 11b. snapshot of my first page of code

a. Rapid Growth in Early Stages

At the beginning, $||w_t||$ grows quickly. I assume this is because in the early stages of PLA, each update brings large adjustments since the algorithm is still far from finding the correct classification boundary.

b. Slowing Growth as Convergence Nears

As the number of updates increases, the growth of $||w_t||$ slows down. This shows PLA is getting closer to the correct classification boundary, where smaller adjustments are needed.

Once the algorithm correctly classifies all samples, the weights top changing, so $||w_t||$ stops growing.

c. Similarity in All Experiments

All curves show a similar pattern of above: rapid growth followed by slower growth and eventually stabilization.

In summary, $||w_t||$ grows quickly at the start of the learning process and then stabilizes as the PLA converges. Also $||w_t||$ shows a similar pattern across different experiments.

12.

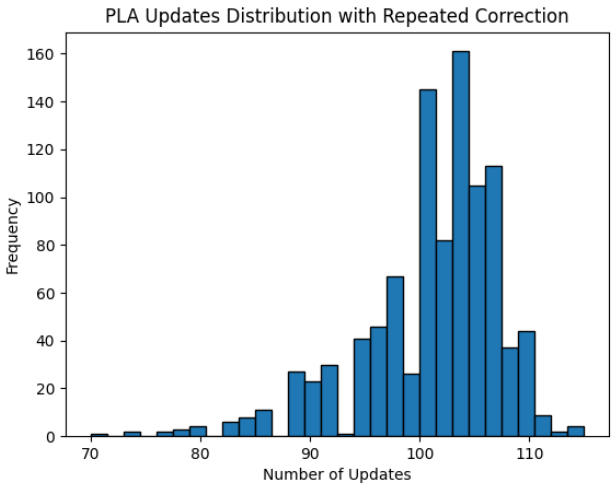

Figure 12a. updates histogram

```python
import numpy as np
import random
from sklearn.datasets import load_svmlight_file
import matplotlib.pyplot as plt

# parameters
experiment = 1000
sample = 200
#max_update = 5000
update_counts = []      # save every experiment update counts

# loading datas
def load_data(path, sample):
    x, y = load_svmlight_file(path)     # loading LIBSVM form
    x = x[:sample].toarray()        # to numpy array
    y = y[:sample]
    return x, y

x, y = load_data('rcv1_train.binary', sample)

# adding bias x0 = 1 to every xn
x =  np.hstack([np.ones((x.shape[0], 1)), x])

# PLA
def pla(x,y):
    w = np.zeros(x[0].shape)       # initialize w = 0
    updates = 0
    sample = x.shape[0]
    max_5N = 5 * sample
    no_error = 0

    while no_error < max_5N:
        i = np.random.choice(sample)   # with replacement
        while np.sign(np.dot(w, x[i])) != y[i]:    # sign different
            w = w + y[i] * x[i]
            updates += 1
        no_error += 1
    return w, updates

for i in range(experiment):
```

Figure 12b. snapshot of my first page of code

```
Median number of updates: 102.0
```

Figure 12c. median number of updates

a. Results Comparison between 10 and 12

In problem 10, we get our median number of updates 102, which is identical to problem 12. Initially, I hypothesized that problem 10 might result in fewer updates overall because it doesn't stick with a wrong sample to fully correct it before moving on. This could leads to a more compact distribution. On the other hand, prob.12 might have a more spread-out update distribution. However, based on the median and the figures, problem 12 only updates once for every sample, which end up with similar update counts and distribution patterns as problem 10.

b. Advantages

By fully correcting each misclassified sample before moving on, the algorithm reduces the chances of revisiting the same problem repeatedly in later iterations. This increase classification accuracy per sample.

It also allows the algorithm to focus on a specific issue and resolve it entirely, which might result in fewer overall updates needed to achieve convergence on difficult samples.

c. Drawbacks

While the algorithm might fully correct each sample, this approach could slow down the overall learning process, as it may spend more time on difficult samples. The trade-off here is between focusing on immediate corrections versus spreading updates more evenly across different samples.

13. Proof 1 :

For a misclassified sample x_n , we have $y_{n(t)}\left(w_t^T x_{n(t)}\right) \leq 0$

From the new update rule, we get

$$w_{t+1} = w_t + \frac{1}{10} y_{n(t)} x_{n(t)} \cdot \lfloor \frac{-10 y_{n(t)} w_t^T x_{n(t)}}{\left||x_{n(t)}\right||^2} + 1 \rfloor \qquad (1)$$

We need to check whether $y_{n(t)}\left(w_{t+1}^T x_{n(t)}\right) > 0$ or not. So, we replace it with (1).

$$y_{n(t)}\left(w_{t+1}^T x_{n(t)}\right) = y_{n(t)}\left(\left(w_t + \frac{1}{10} y_{n(t)} x_{n(t)} \cdot \left\lfloor \frac{-10 y_{n(t)} w_t^T x_{n(t)}}{\left||x_{n(t)}\right||^2} + 1 \right\rfloor\right)^T x_{n(t)}\right)$$

$$= y_{n(t)} w_t^T x_{n(t)} + \left(\frac{1}{10} y_{n(t)}^2 x_{n(t)} \cdot \left\lfloor \frac{-10 y_{n(t)} w_t^T x_{n(t)}}{\left||x_{n(t)}\right||^2} + 1 \right\rfloor\right) x_{n(t)}$$

Since $y_{n(t)}^2 = 1$, we get

$$y_{n(t)} w_t^T x_{n(t)} + [x_{n(t)} \cdot \left\lfloor \frac{-y_{n(t)} w_t^T x_{n(t)}}{\left||x_{n(t)}\right||^2} + \frac{1}{10} \right\rfloor] x_{n(t)}$$

$$= y_{n(t)} w_t^T x_{n(t)} + \left||x_{n(t)}\right||^2 \cdot \left\lfloor \frac{-y_{n(t)} w_t^T x_{n(t)}}{\left||x_{n(t)}\right||^2} + \frac{1}{10} \right\rfloor$$

$$= y_{n(t)} w_t^T x_{n(t)} - y_{n(t)} w_t^T x_{n(t)} + \frac{\left||x_{n(t)}\right||^2}{10} = \frac{\left||x_{n(t)}\right||^2}{10}$$

Since $\left||x_{n(t)}\right||^2$ is always positive, $y_{n(t)}\left(w_{t+1}^T x_{n(t)}\right)$ ends in positive. The updated weight vector will always correctly classify after the update.

Proof 2 :