HW1

chee han

September 26, 2024

1 Problem Description

Ask any ChatGPT-like agent the following question: *"What is a possible application of active learning?"*, list the answer that you get, and argue with 10-20 English sentences on whether you agree with the agent or not, as if you are the "boss" of the agent.

The TAs will grade based on the persuasiveness of your arguments—please note that our TAs are more used to being persuaded by humans than machines. So if your arguments do not look very human-written, the TAs may not be persuaded.

1.1 CHATGPT

Application of Active Learning in Algorithmic Trading In algorithmic trading, traders rely on models to predict price movements or generate trading signals. However, training these models requires large amounts of labeled data, such as identifying patterns that lead to profitable trades. Labeling such data can be challenging due to the complexity of the market and the need for expert insights.

With active learning, the model can intelligently select the most ambiguous or uncertain data points from market data (e.g., historical price data, technical indicators).

1.2 My Opinion

I agree that active learning could be applied in algorithmic trading, but there are some reasons why I don't agree with active learning being used for predicting prices. For instance, most data types are non-stationary (e.g., price data), meaning that historical data may no longer be useful for predicting future prices.

In my opinion, active learning could be used to predict future returns, because it has more stable statistical properties and tends to be more normally distributed.

2 Problem Description

Go ask any chatGPT-like agent the following question, "can machine learning be used to predict earthquakes?", list the answer that you get, and argue with 10-20 English sentences on whether you agree with the agent or not, as if you are the "boss" of the agent.The TAs will grade based on the persuasiveness of your arguments—please note that our TAs are more used to being persuaded by humans than machines. So if your arguments do not look very human-written, the TAs may not be persuaded.

2.1 CHATGPT

Machine learning can be used to help predict earthquakes, although accurately predicting the exact time, location, and magnitude of an earthquake remains a significant challenge.

2.2 My Opinion

In my opinion, I would agree that predicting earthquakes is still a challenging task. First, the factors that cause earthquakes are complex, making it difficult to determine which factors have a more significant impact. This complexity poses a challenge for machine learning because a highly precise prediction requires high-quality data. Lastly, I still believe that predicting earthquakes may become possible in the future because the problem has a clear definition.

3 Problem Description

Before running PLA, our class convention adds $x_0 = 1$ to every x_n vector, forming $x_n = (1, x_{\text{orig},n})$. Suppose that $x'_0 = 2$ is added instead to form $x'_n = (2, x_{\text{orig},n})$.

Consider running PLA on $\{(x_n, y_n)\}_{n=1}^N$ in a cyclic manner with the naïve cycle. That is, the algorithm keeps finding the next mistake in the order of $1, 2, \ldots, n, 1, 2, \ldots$. Assume that such a PLA with $w_0 = 0$ returns w_{PLA}, and running PLA on $\{(x'_n, y_n)\}_{n=1}^N$ with the same cyclic manner with $w_0 = 0$ returns w'_{PLA}.

Prove or disprove that w_{PLA} and w'_{PLA} are equivalent. We define two weight vectors to be equivalent if they return the same binary classification output on every possible example.

3.1 Answer

- Original feature vector: $x_n^{\text{orig}} \in \mathbb{R}^d$.
- Augmented vectors:
  - $x_n = (1, x_n^{\text{orig}}) \in \mathbb{R}^{d+1}$.

- $x'_n = (2, x_n^{\text{orig}}) \in \mathbb{R}^{d+1}$.

- Weight vectors:
  - $w = (w_0, w_{\text{q}}) \in \mathbb{R}^{d+1}$.
  - $w' = (w'_0, w'_{\text{q}}) \in \mathbb{R}^{d+1}$.
- Labels: $y_n \in \{-1, +1\}$.

1. Bias term update :
   - The update to the bias term w_0 when a misclassificate occurs is:
   $$w_0 \leftarrow w_0 + y_n \cdot 1 = w_0 + y_n.$$
   - The update to the bias term w'_0 is:
   $$w'_0 \leftarrow w'_0 + y_n \cdot 2 = w'_0 + 2y_n.$$
2. The updates to w_{q} and w'_{q} (rest) are :
   $$w_{\text{q}} \leftarrow w_{\text{q}} + y_n x_n^{\text{orig}}.$$
   $$w'_{\text{q}} \leftarrow w'_{\text{q}} + y_n x_n^{\text{orig}}.$$
3. We have :
   $$w'_0 = 2w_0.$$
   because if $w_0 = 0$ and $w'_0 = 0$ at the start, then:
   - First, $w_0 + 1 = 1$ and $w'_0 + 2 = 2$.
   - Next, $w_0 + 1 = 2$ and $w'_0 + 2 = 4$.

   This pattern continues again and again.
   - For the original:
   $$\text{sign}(w \cdot x_n) = \text{sign}(w_0 + w_{\text{q}} \cdot x_n^{\text{orig}}).$$
   - For the modified:
   $$\text{sign}(w' \cdot x'_n) = \text{sign}(w'_0 \cdot 2 + w'_{\text{q}} \cdot x_n^{\text{orig}}).$$
   Substituting $w'_0 = 2w_0$ and $w'_{\text{q}} = w_{\text{q}}$:
   $$\text{sign}(w' \cdot x'_n) = \text{sign}(2 \cdot 2w_0 + w_{\text{q}} \cdot x_n^{\text{orig}}) = \text{sign}(4w_0 + w_{\text{q}} \cdot x_n^{\text{orig}}).$$
4. **Counterexample:** Let:
   - $w_0 = -1$.
   - $w_{\text{q}} \cdot x_n^{\text{orig}} = 1$.(assume a x in R^{d} space.)

Then:

- Standard PLA decision:

$$D(w) = (-1) + 1 = 0.$$

$$\text{sign}(D(w)) = 0.$$

- Modified PLA decision:

$$D'(w') = 4(-1) + 1 = -4 + 1 = -3.$$

$$\text{sign}(D'(w')) = -1.$$

5. **Disproved!**: w_{PLA} and w'_{PLA} are not equivalent, they can yield different classifications on some inputs.

4 Problem Description

Before running PLA, our class convention adds $x_0 = 1$ to every x_n vector, forming $x_n = (1, x_{\text{orig},n})$. Suppose that we scale every x_n by 3, and $x'_0 = 3$ is added instead to form $x'_n = (3, 3x_{\text{orig},n})$.

Consider running PLA on $\{(x_n, y_n)\}_{n=1}^N$ in a cyclic manner with the naïve cycle. That is, the algorithm keeps finding the next mistake in the order of $1, 2, \ldots, n, 1, 2, \ldots$. Assume that such a PLA with $w_0 = 0$ returns w_{PLA}, and running PLA on $\{(x'_n, y_n)\}_{n=1}^N$ with the same cyclic manner with $w_0 = 0$ returns w'_{PLA}.

Prove or disprove that w_{PLA} and w'_{PLA} are equivalent.

4.1 Answer

the update rule becomes:

$$\begin{aligned} w'_{t+1} &= w'_t + y_n x'_n \\ &= w'_t + y_n(3x_n) \\ &= w'_t + 3y_n x_n. \end{aligned}$$

I am going to use mathematical induction to show that $w'_t = 3w_t$ at each iteration t.

- **Base Case** ($t = 0$):
  We assume $w_0 = 0$ and $w'_0 = 0$. Therefore,

$$w'_0 = 3w_0 = 3 \times 0 = 0.$$

- **Inductive Step**:
  Assume $w'_t = 3w_t$ holds for some t. We need to show that $w'_{t+1} = 3w_{t+1}$.

  From the PLA update rule:

  $$w_{t+1} = w_t + y_n x_n.$$

  Using the inductive hypothesis $w'_t = 3w_t$, the update for the scaled dataset is:

  $$\begin{aligned} w'_{t+1} &= w'_t + 3y_n x_n \\ &= 3w_t + 3y_n x_n \\ &= 3(w_t + y_n x_n) \\ &= 3w_{t+1}. \end{aligned}$$

  Thus, $w'_{t+1} = 3w_{t+1}$, completing the induction.

Since the weight vectors are scalar multiples of each other, they define the same decision boundary (the sign of the dot product $w^\top x$ remains the same). Therefore, w_{PLA} and w'_{PLA} are equivalent in terms of classification.

4.2 Prove!

Scaling the input vectors by a positive scalar results in the weight vector being scaled by the same factor after running PLA with the same initialization and update order(just a data scaling in proportional). Therefore, we can say that the final weight vectors are equivalent.In conclusion ,running PLA on the scaled data scales the weight vector by 3, so $w'_{\text{PLA}} = 3w_{\text{PLA}}$.

9. From the problem description we know that it is a bag-of words representation, so it is binary representation for data set.

By converge theorem we know that the number of mistake is bounded by

$$T \le \left(\frac{R}{\rho}\right)^2$$

$\rho = \min \frac{y_n w_t^T x_n}{\|w_t\|}$, which threshold is 3.5, so min is $|4-3.5|, |3-3.5|$

$= \frac{0.5}{\|w_t\|}$

$R = \max \|x_n\|$, since $d \ge m$

$= \sqrt{1+m}$, 1 is w0

$R^2 = (1+m)^2$

$\|w_t\|^2 = \|w_0\|^2 + \sum_{i=1}^{d} \|w_i\|^2$, we assume the loop will give the w by -1 or +1, if hatred +1 if no -1.

$= (3.5)^2 + d$

$= 12.25 + d$

since, $T \le \left(\frac{R}{\rho}\right)^2$

$$T \le \frac{(1+m)(12.25+d)}{0.5^2}$$

$$\le \frac{(1+m)(12.25+d)}{\frac{1}{4}}$$

$$T \le 4(1+m)(12.25+d) \#$$

10.

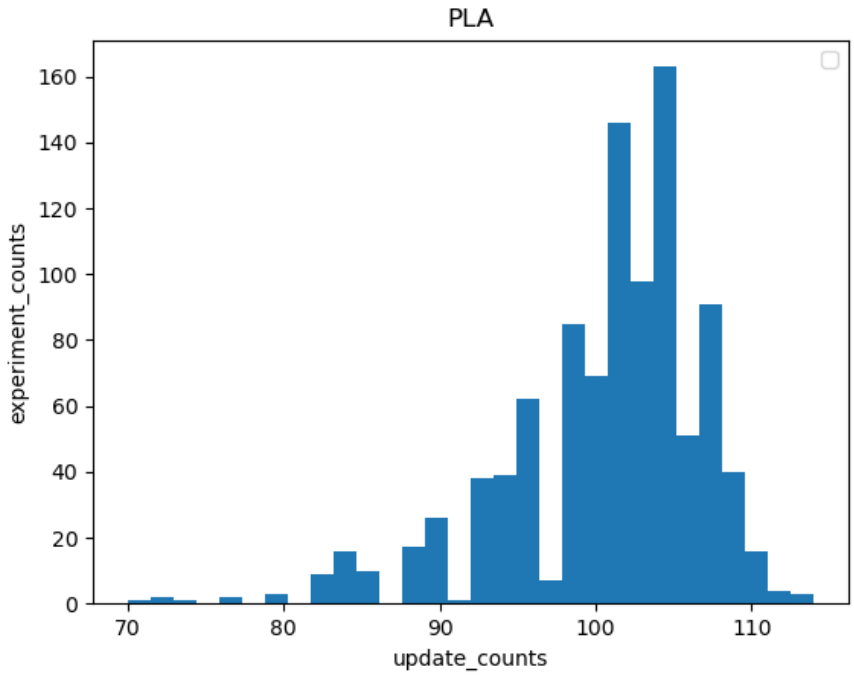
Graph Explain： The following shows the results of 1000 attempts of the PLA. The x-axis represents the range of updates required for the algorithm to converge, while the y-axis indicates how many experiments converged within that range.

observe：

I． From the graph, it can be observed that the number of updates mostly falls within the range of (100-110) updates before full convergence. This suggests that the convergence speed of the samples typically falls within this range, with a chance of achieving convergence with fewer updates.

II． The histogram shows a shape close to a normal distribution, indicating that the number of updates is influenced by randomness, consistent with the Central Limit Theorem.

III． Initially, the idea was to generate a space of w*47206, but after a deeper study of sparse matrices, the algorithm was changed to only generate matrices corresponding to the indices of x ,which significantly improved the speed.



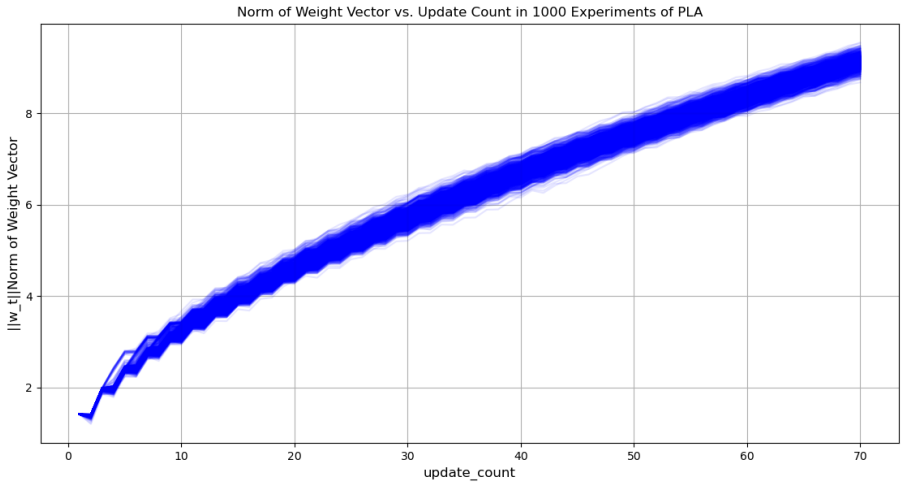
```
    return y , x
def dot_product(w, x_i):
    dot   =  0
    for index, key_value in x_i.items() :
        init_w = w.get(index, 0.0)
        dot    += init_w*key_value

    return dot

def PLA(x, N, y) :
    w = {}
    correct_count = 0
    update_count = 0
    while correct_count < 5*N :
        indx = random.randint(0,N-1)
        y_i  = y[indx]
        x_i  = x[indx]
        dot  = dot_product(w, x_i)
        predict = predictor_sign(dot)
        if predict != y_i :
            weight_update (w, x_i, y_i)
            correct_count = 0
            update_count += 1
        elif predict == y_i :
            correct_count += 1
```

```
def weight_update(w , x_i , y_i):
    for index , item in x_i.items() :
        w[index] = w.get(index, 0.0) + y_i*item



def predictor_sign(dot) :
    if dot > 0:
        return 1
    if dot <= 0 :
        return -1


def main() :
    update_counts = []
    N     = 200
    y , x  = preprocess_data('rcv1_train.txt',N)

    for count in range(1000):
        random.seed(count)
        count = PLA( x, N, y)
```

11. Graph Description: Below is the plot of update count versus the normalized weight vector $w_{\circ}$

I. Each update is relatively small, indicating that the learning process of the PLA algorithm is gradual rather than abrupt.

II. From the experimental results, although the weight norm curves in different experiments show slight variations, the overall trend remains consistent, indicating that the convergence process of the PLA is not significantly affected by the random order of sample selection.




```python
def calculate_normalize(w):

    w_values = np.array(list(w.values()))

    return np.linalg.norm(w_values)


def dot_product(w, x_i):
    dot = 0
    for index, key_value in x_i.items():
        init_w = w.get(index, 0.0)
        dot += init_w * key_value
    return dot
```

```python
def PLA(x, N, y):
    w = {}
    correct_count = 0
    update_count = 0
    w_path = []
    while correct_count < 5 * N:
        indx = random.randint(0, N - 1)
        y_i = y[indx]
        x_i = x[indx]
        dot = dot_product(w, x_i)
        predict = predictor_sign(dot)
        if predict != y_i:
            weight_update(w, x_i, y_i)
            correct_count = 0
            update_count += 1
            norm = calculate_normalize(w)
            w_path.append(norm)
        else:
            correct_count += 1

    return update_count, w_path
```

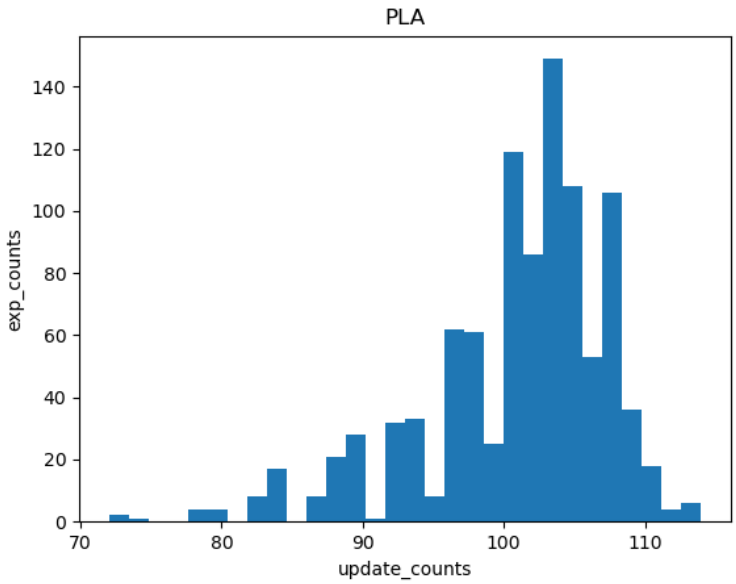
```python
def main():
    update_counts = []
    path_exp = []
    N = 200
    y, x = preprocess_data('rcv1_train.txt', N)

    for trial in range(1000):
        random.seed(trial)
        count, w_path = PLA(x, N, y)
        update_counts.append(count)
        path_exp.append(w_path)


    T_min = min(update_counts)
    print(f"T_min: {T_min}")

    truncated_norms = [norms[:T_min] for norms in path_exp]


    t_values = list(range(1, T_min + 1))

    plt.figure(figsize=(10, 6))

    for norms in truncated_norms:
        plt.plot(t_values, norms, color='blue', alpha=0.1)

    plt.xlabel('update_count', fontsize=12)
    plt.ylabel('||w_t||Norm of Weight Vector', fontsize=12)
    plt.title('Norm of Weight Vector vs. Update Count in 1000 Experiments of PLA', fontsize=14)
    plt.grid(True)
    plt.show()
```

12.

Graph description:

median : 102.0

I. From the graph, I saw that using the same point for updates (Picture No.2) is better than always using a random point for updates (Picture No.1). In my opinion, I think using the same point for updates works better when the data has less noise.




```python
def PLA(x, N, y) :
    w = {}
    correct_count = 0
    update_count = 0
    indx = random.randint(1,N-1)
    while correct_count < 5*N :

        y_i  = y[indx]
        x_i  = x[indx]
        dot  = dot_product(w, x_i)
        predict = predictor_sign(dot)
        if predict != y_i :
            weight_update (w, x_i, y_i)
            correct_count = 0
            update_count += 1
        elif predict == y_i :
            correct_count += 1
            indx = random.randint(0, N - 1)
    return update_count
```