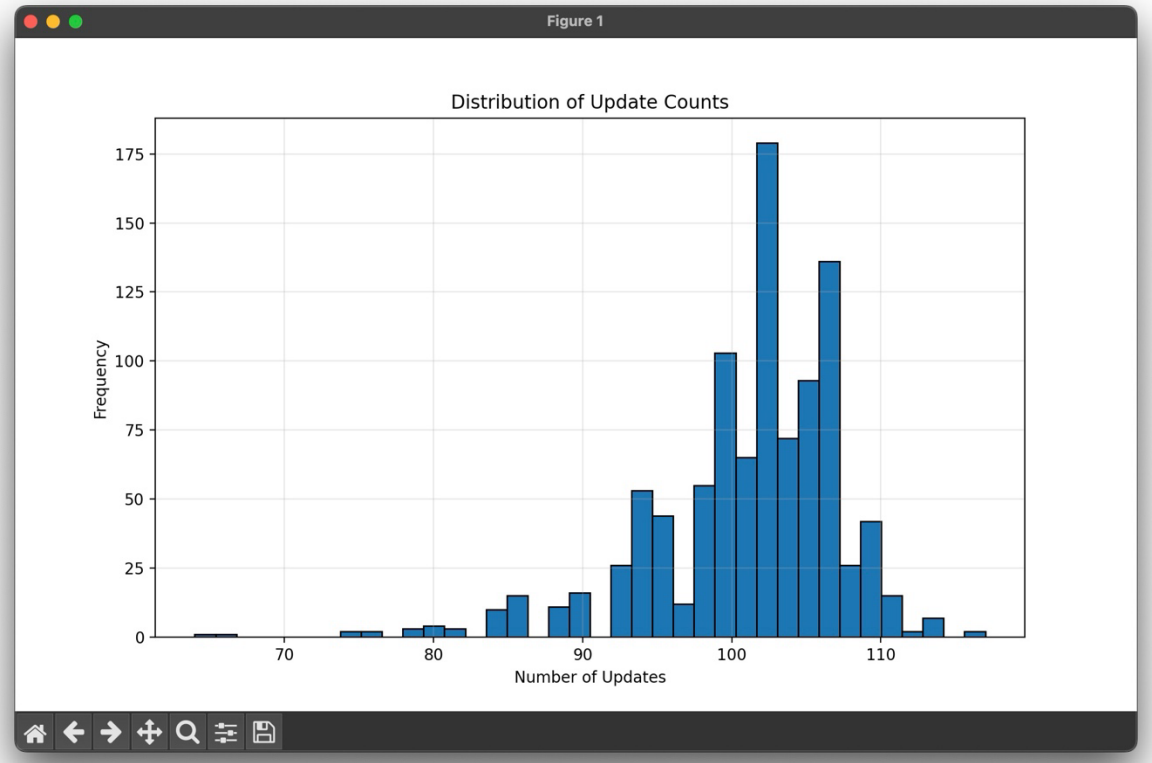
ML homework 1: Question 10

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:

```
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
```

```
52    for index, line in enumerate(lines):
53        if index >= N:
54            break
55
56        parts = line.split()
57        label = int(parts[0])
58        y_arr.append(1 if label == 1 else -1)
59        sparse_vector = ["0:1"] + parts[1:]
60        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:

```
91        for random_no in random_check_indices:
92
93            sparse_vector = sparse_vector_arr[random_no]
94            label = y_arr[random_no]
95            sign = dot_product_sign(sparse_vector, w)
96            if sign != label:
97                for item in sparse_vector:
98                    index, value = item.split(':')
99                    index = int(index)
100                   w[index] += label * float(value)
101               correct_amount = 0
102               current_update_times += 1
103           else:
104               correct_amount += 1
105
106
107           if correct_amount == 1000:
108               break
109
110           # if the current update times is a multiple of 1000, this means that the current "random_check_indices" array has used out,
111           # so we update the array with 1000 new random indices
112           if current_update_times % 1000 == 0:
113               random_check_indices = [next(random_sequence) for _ in range(5 * N)]
114
115   update_times_arr.append(current_update_times)
```