1.

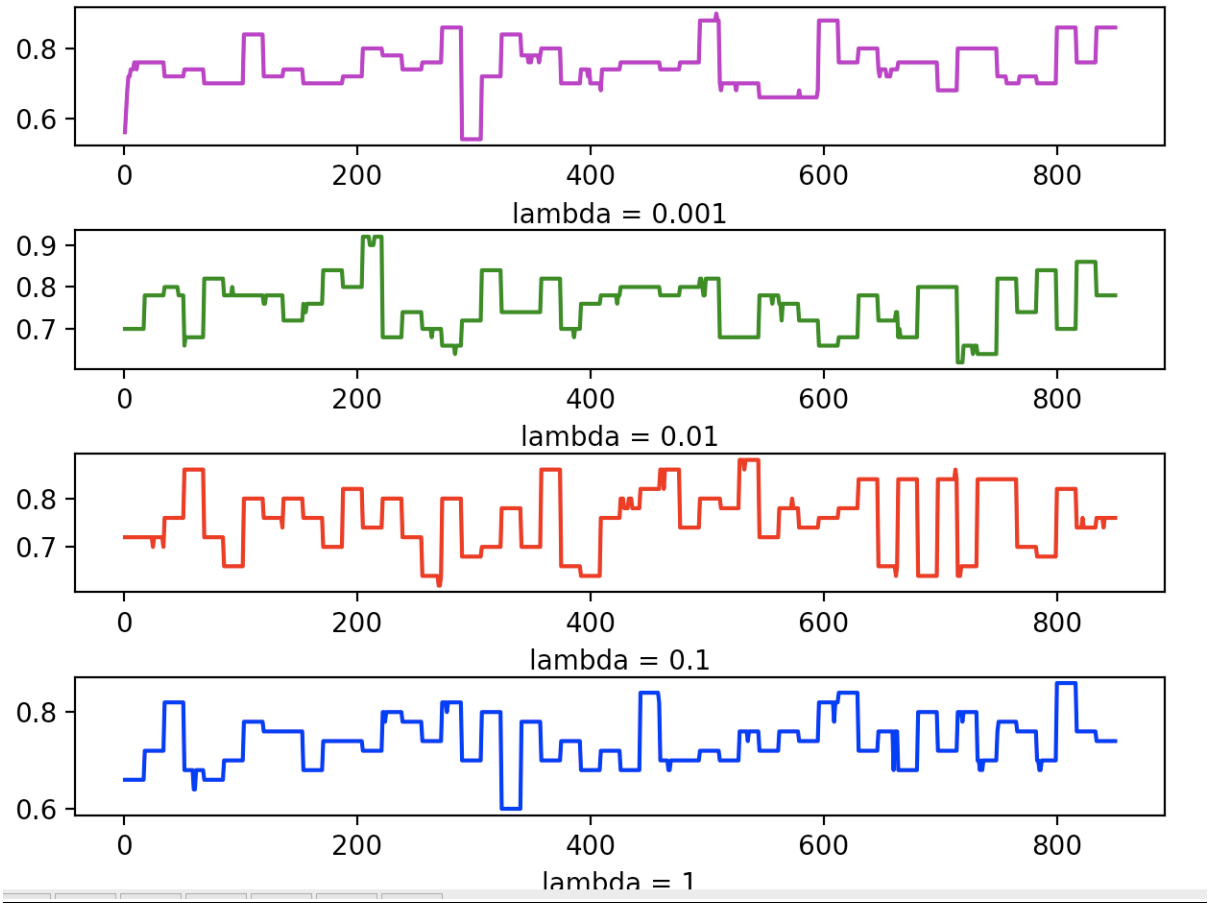
94	new	**Chongye Wang**		0.76760	22	now

Your Best Entry ↑

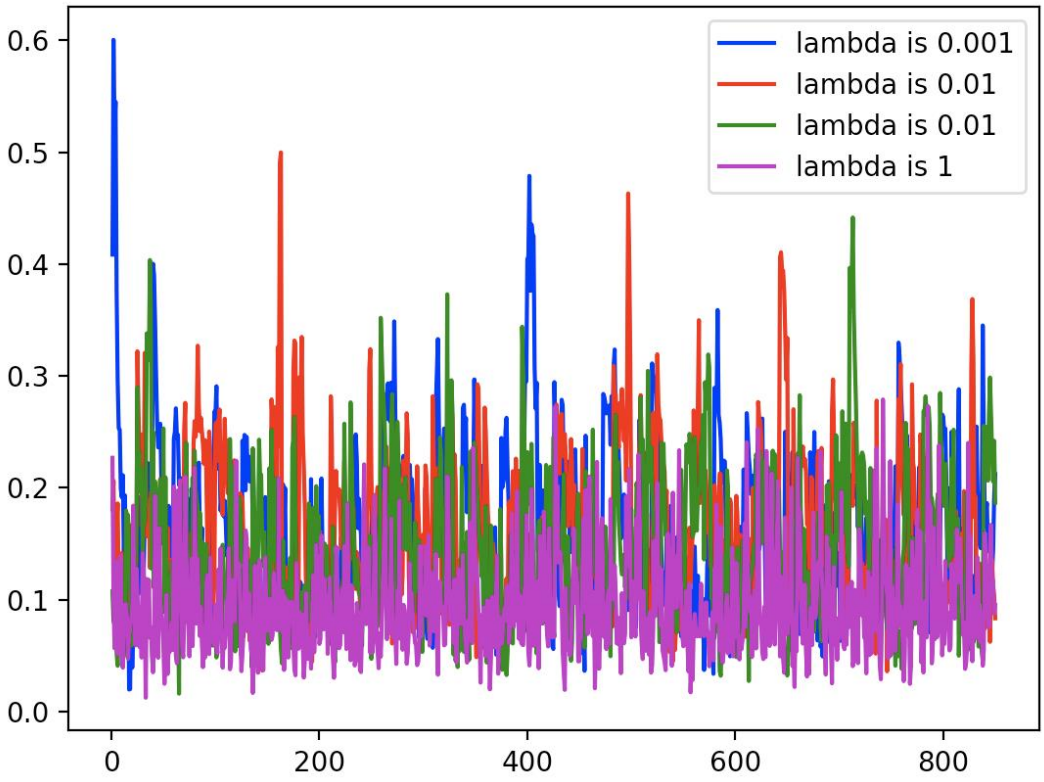
You advanced 7 places on the leaderboard!

Your submission scored 0.76760, which is an improvement of your previous score of 0.58538. Great job! Tweet this!

2.



3.



4.
Lambda : 0.001
After running, we can see that compared with other values, 0.001 shows a more stable result after several epochs and shows a relatively higher accuracy and more stable curve. Therefore, I choose lambda = 0.001

m = 1
n = 50

learning rate = m / (0.01 * epoch + n)

I my choice, the learning rate will have a stable decrease each time as the epoch increases but after multiplication with 0.01 the learning rate will not decrease too sharply.

5.

```
for epoch in range(50):

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)

    if epoch == 49:
        result = get_accuracy(a, b, X_test, y_test)
        print(str(lamb) + ' : ' + str(result))

    shuffle(X_train)

    validation_train = X_train[0:50]
    validation_test = y_train[0:50]

    train_data = X_train[51:]
    train_test = y_train[51:]

    m = 1
    n = 50
    step_size = m / (0.01 * epoch + n)

    for step in range(500):

        if step % 30 == 0:
            accuracy = get_accuracy(a, b, validation_train, validation_test)

            dict_accuracy[lamb].append(accuracy)
            dict_a[lamb].append(a)
            dict_b[lamb].append(b)

        # current index randomly chosen
        curr = random.randint(0, len(train_data))

        curr_train = np.array(train_data[curr])

        curr_train = curr_train.reshape(1, 6)

        curr_val = (curr_train.dot(a.T) + b) * train_test[curr]

        if curr_val >= 1:
            a = a - np.dot(a, lamb) * step_size
        else:
            a = a - step_size * (np.dot(a, lamb) - np.dot(train_data[curr], train_test[curr]))
```