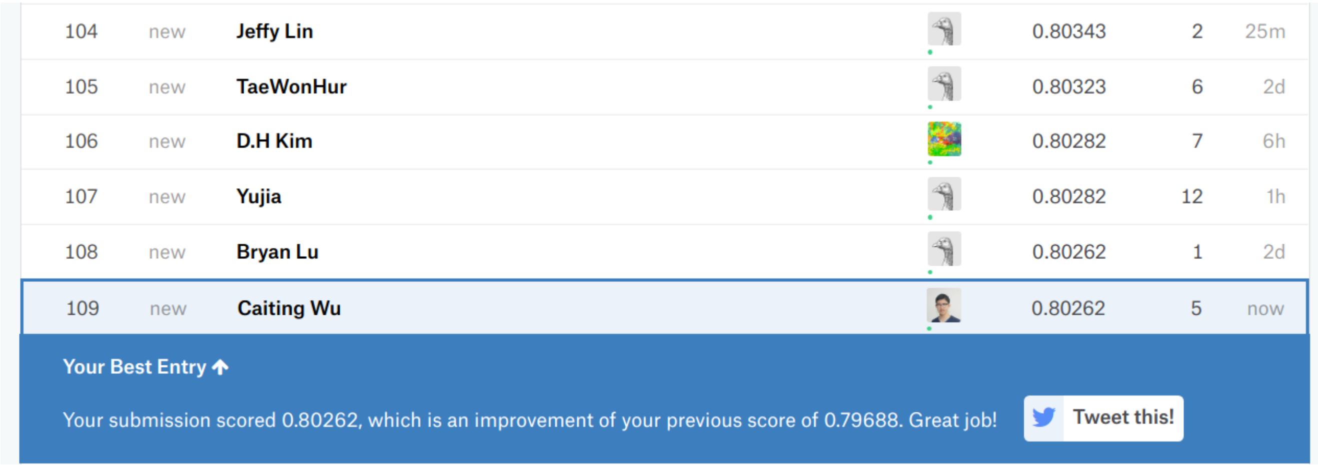
CS 498 AML Homework 2 Report Caitng Wu

1. Screenshot of leader-board accuracy:



104	new	Jeffy Lin	0.80343	2	25m
105	new	TaeWonHur	0.80323	6	2d
106	new	D.H Kim	0.80282	7	6h
107	new	Yujia	0.80282	12	1h
108	new	Bryan Lu	0.80262	1	2d
109	new	Caiting Wu	0.80262	5	now

Your Best Entry

Your submission scored 0.80262, which is an improvement of your previous score of 0.79688. Great job!

Tweet this!

The best test data accuracy that I have obtained on Kaggle is: 0.80262

2. A plot of the accuracy every 30 steps:

(epochs = 50, steps/echo = 300, batch size = 160 [last batch < 160])

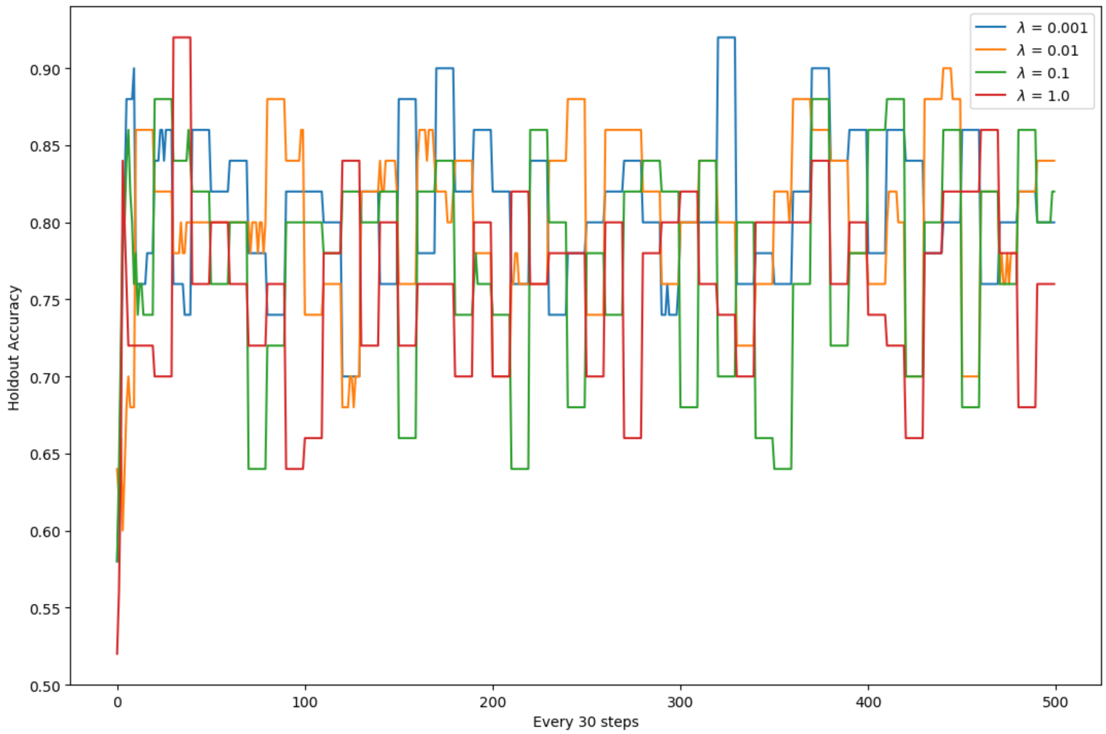


figure1: The plot of Holdout accuracy over every 30 steps with four different λ values

3. A plot of the magnitude of the coefficient vector every 30 steps:

(epochs = 50, steps/echo = 300, batch size = 160 [last batch < 160])

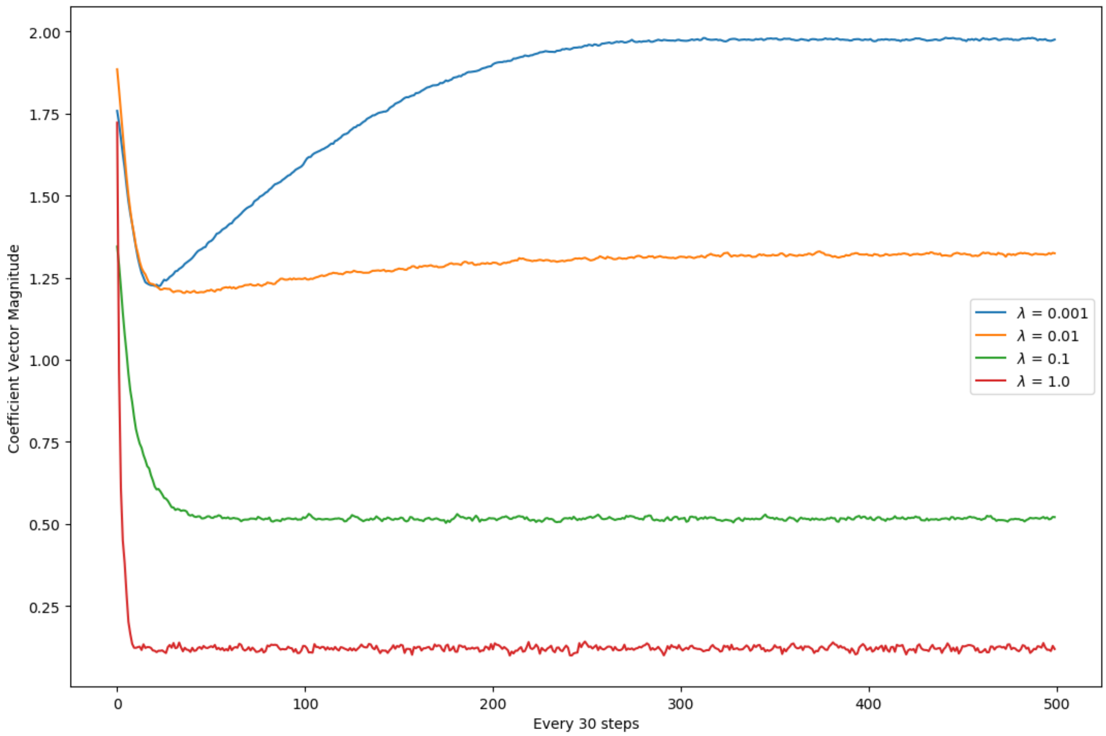


figure2: The plot of coefficient magnitude over every 30 steps with four different λ values

4. Reasoning the best regularization constant and learning rate:

a. My estimation of the best regularization constant is 0.01:

~ From the accuracy graph, we can conclude that the validation accuracy does not vary too much among different regularization constants. When $\lambda=0.01$ and $\lambda=0.001$, we tend to receive good validation results.

~ From the running report, we can see that the best accuracy that the model with these two constants could achieve are no significantly different from each other.

```
In [13]: # Showing the parameters and their results
    ...: for key, value in lamb_best_config.items():
    ...:     print('With lambda = ', key, ' the best accuracy is: ', value['acc'])
    ...:
With lambda =  0.0002  the best accuracy is:  0.8123293903548681
With lambda =  0.0001  the best accuracy is:  0.8123293903548681
With lambda =  0.002  the best accuracy is:  0.8166515013648772
With lambda =  0.001  the best accuracy is:  0.8123293903548681
With lambda =  0.01  the best accuracy is:  0.8105095541401274
With lambda =  0.1  the best accuracy is:  0.798680618744313
With lambda =  1.0  the best accuracy is:  0.77024567788899
```

~ During Kaggle course contest submission, $\lambda=0.01$ yielded the best testing result.

b. My estimation of the learning rate is $\frac{1}{0.01\times epoch+50}$:

During the training, I found that modifying the learning rate provided in the textbook does not offer significant improvement. The learning rate that I have tried are: $\frac{1}{0.01\times epoch+50}$, $\frac{2}{0.01\times epoch+50}$ and $\frac{1}{0.01\times epoch+5}$. The result confirms with what mentioned in the textbook that the step-length tend to explore large changes in parameters and settle down afterwards.

5. Codes:

```
def svm_sgd (epochs, lam, tol_steps, train_x, train_y, val_x, val_y):
    mag_list = []
    stepacc_list = []
    best_config = {'acc': 0.0, 'a':0, 'b': 0}
    index_array = np.array(range(len(train_x)))

    a = np.random.random(train_x.shape[1])
    b = np.random.random(1)[0]

    for each in range(epochs):
        step_len = 1/(0.01*each + 50)
        # shuffle the entire data set at each epoch
        np.random.shuffle(index_array)
        hold_index = index_array[-50:]
        # split the data index array into a almost evenly splited array
        batch_index = np.array_split(index_array[:-50], tol_steps)

        for step in range(tol_steps):
            # Extract current minibatch data and labels
            x_k = train_x[batch_index[step], :]
            y_k = train_y[batch_index[step]]
            ax_b = x_k.dot(a) + b
            temp = y_k*ax_b
            big_idx = np.where(temp >= 1)[0]
            small_idx = np.where(temp < 1)[0]
            if len(big_idx) + len(small_idx) != len(temp):
                print("The length of separating indeces is wrong!!")
            delta_a = len(big_idx)*step_len*lam*a + step_len*(len(small_idx)*lam*a - y_k[small_idx].T.dot(x_k[small_idx, ]))
            delta_b = (-1)*step_len*np.sum(y_k[small_idx])
            # Updating a and b
            a -= 1./len(batch_index[step])*delta_a
            b -= 1./len(batch_index[step])*delta_b

            if (step%30 == 0):
                curr_acc, _, _ = test_acc(train_x[hold_index], train_y[hold_index], a, b)
                a_norm = np.linalg.norm(a)
                stepacc_list.append(curr_acc)
                mag_list.append(a_norm)
                if val_x == None and val_y == None:
                    if curr_acc > best_config['acc']:
                        best_config['a'] = a
                        best_config['b'] = b
                        best_config['acc'] = curr_acc

        if val_x != None and val_y != None:
            valid_acc, predict, result = test_acc(val_x, val_y, a, b)
            if valid_acc > best_config['acc']:
                best_config['a'] = a
                best_config['b'] = b
                best_config['acc'] = valid_acc

    return mag_list, stepacc_list, best_config
```

```
def make_predict(feature: np.ndarray, a: np.ndarray, b: int):
    predict = feature.dot(a) + b
    predict[predict > 0] = 1
    predict[predict <= 0] = -1
    return predict
```

```
def test_acc(feature: np.ndarray, label: np.ndarray, a: np.ndarray, b: int):
    predict = feature.dot(a) + b
    predict[predict > 0] = 1
    predict[predict <= 0] = -1
    if(label.shape[0] != predict.shape[0]):
        print('Something is wrong with the prediction array size.\n')
    result = predict + label
    acc = 1 - (1.*np.where(result == 0)[0].shape[0] / len(result))
    return acc, predict, result
```