CS 498 AML HW2 REPORT
Pengyu Cheng
pcheng11

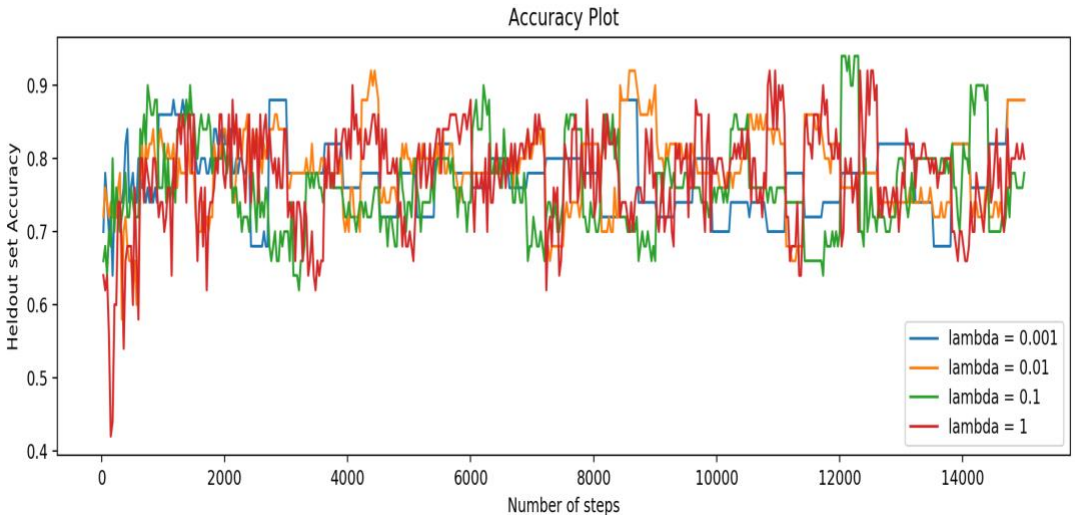
- My leaderboard screenshot

43	new	**95chenjz**	0.80733	7	19h
44	new	**Meishan Wu**	0.80733	3	11h
45	new	**test_account**	0.80692	16	3d
46	new	**Pan Zhang**	0.80692	5	1d
47	new	**Shuyue Lai**	0.80692	22	1d
48	new	**JoJo**	0.80671	18	12h
49	new	**yanxu**	0.80671	16	29m
50	new	**Sean**	0.80671	1	2m
51	new	**PengyuCheng**	0.80610	38	2m
Your Best Entry ↑					
Your submission scored 0.80343, which is not an improvement of your best score. Keep trying!					
52	new	**YuxuanRen**	0.80610	3	21h
53	new	**Lynn**	0.80610	8	13h
54	new	**Savya Saachi Verma**	0.80589	3	2d
55	new	**Allen Tang**	0.80589	1	18h
56	new	**Chen Pan**	0.80589	2	13h

- My best test dataset accuracy obtained on kaggle is 0.80610

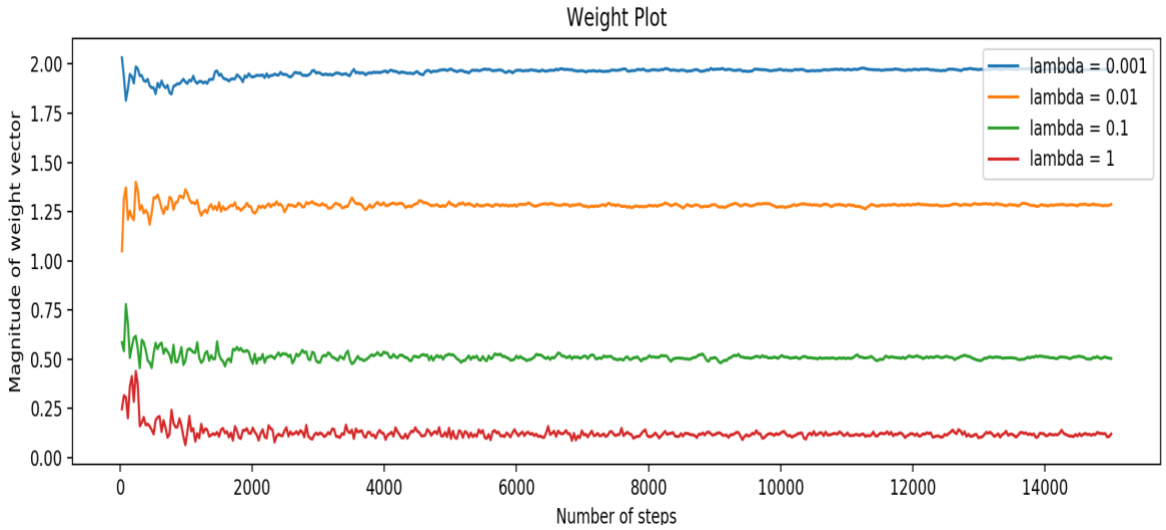
CS 498 AML HW2 REPORT
Pengyu Cheng
pcheng11

- A plot of the accuracy every 30 steps, for each value of the regularization constant. (epochs = 50, steps = 300, batch size = 131)



CS 498 AML HW2 REPORT
Pengyu Cheng
pcheng11

- A plot of the magnitude of the coefficient vector every 30 steps, for each value of the regularization constant.
  (epochs = 50, steps = 300, batch size = 131)



CS 498 AML HW2 REPORT
Pengyu Cheng
pcheng11

My estimate of the best regularization constant is 0.01.

- We can see from the accuracy graph that the accuracy is not particularly sensitive to the regularization constant but tend to give a good result when lambda is 0.01.

- The graph below also testifies the choice of my lambda.

```
2018-09-17 14:10:41,173 - SVM Model - INFO - Preprocessing data...
2018-09-17 14:10:41,447 - SVM Model - INFO - Preprocessing done!
2018-09-17 14:10:41,447 - SVM Model - INFO - Train set size: 39562, val set size: 4396, test set size: 4884
2018-09-17 14:10:41,447 - SVM Model - INFO - Epochs = 50, Steps = 300, Batch size = 131
2018-09-17 14:10:41,447 - SVM Model - INFO - Searching best regularization constant lambda...
100%|
2018-09-17 14:10:47,432 - SVM Model - INFO - Accuracy for lambda = 0.001 is 0.7909463148316651
100%|
2018-09-17 14:10:53,919 - SVM Model - INFO - Accuracy for lambda = 0.01 is 0.8000454959053686
100%|
2018-09-17 14:11:00,756 - SVM Model - INFO - Accuracy for lambda = 0.1 is 0.7836669699727025
100%|
2018-09-17 14:11:07,458 - SVM Model - INFO - Accuracy for lambda = 1 is 0.781164695177434
2018-09-17 14:11:07,458 - SVM Model - INFO - Searching done! lambda = 0.01
```

We see that when the lambda = 0.01, the accuracy of the validation set is around 80% which is the largest value.

My choice of learning rate is 1/ (number of epochs).

- I have tried a several of learning rates either constant = 0.01 or diminishing learning rate such as 1/(number of epochs + 50) as given in the book but my chosen learning rate of this specific lambda and batch size tends to give a good result. From another point of view, just as the books says: “that the method can explore large changes in the values of the classifier parameters — and small steps later — so that it settles down”.

CS 498 AML HW2 REPORT
Pengyu Cheng
pcheng11

```python
def split_train_val(train_set):
    train_set, val_set = train_test_split(train_set, test_size=0.1)
    return train_set, val_set

def initialize(mu, sigma):
    w = np.random.normal(mu, sigma, 6)
    b = np.random.normal(mu, sigma, 1)
    return w, b

def generate_mini_batch(train_set, batch_size):
    for batch_start in range(0, len(train_set), batch_size):
        yield [train_set[batch_start: batch_start+batch_size, :-1], train_set[batch_start: batch_start+batch_size,-1]]

def train_svm(epochs, steps, batch_size, train_set, lbda, logger):
    accuracy_list = []
    weight_list = []
    step_list = []
    w, b = initialize(0, 1)
    for epoch in tqdm(range(0, epochs)):
        np.random.shuffle(train_set)
        heldout_set = train_set[:50, :]
        sub_train_set = train_set[50:, :]
        learning_rate = 1/(epoch+1)
        generator = generate_mini_batch(sub_train_set, batch_size)
        for step in range(1, steps+1):
            try:
                x, y = next(generator)
                w, b = update(learning_rate, w, b, lbda, x, y, batch_size)
            except StopIteration:
                continue
            if step%30 == 0:
                accuracy_list.append(validate(heldout_set, w, b))
                weight_list.append(np.linalg.norm(w))
                step_list.append(epoch*steps + step)
    logger.info('Avg accuracy for heldout set is {}'.format(np.mean(accuracy_list)))
    return w, b, accuracy_list, weight_list, step_list
```

```python
def find_lbda(epochs, steps, batch_size, train_set, val_set, lbdas, logger):
    chosen_lbda = None
    cur_max_accuracy = 0
    for lbda in lbdas:
        w, b = initialize(0, 1)
        for epoch in tqdm(range(0, epochs)):
            np.random.shuffle(train_set)
            generator = generate_mini_batch(train_set, batch_size)
            learning_rate = 1/(epoch+1)
            for step in range(0, steps):
                try:
                    x, y = next(generator)
                    w, b = update(learning_rate, w, b, lbda, x, y, batch_size)
                except StopIteration:
                    continue
        accuracy = validate(val_set, w, b)
        logger.info('Accuracy for lambda = {0} is {1}'.format(lbda, accuracy))
        if accuracy >= cur_max_accuracy:
            chosen_lbda = lbda
            cur_max_accuracy = accuracy
    return chosen_lbda


def normalize_split(train_set, test_set):
    for i in range(len(train_set[0])-1):
        col_mean_train = np.mean(train_set[:,i])
        col_std_train = np.std(train_set[:,i])
        train_set[:, i] = (train_set[:, i] - col_mean_train)/col_std_train
        test_set[:, i] = (test_set[:, i] - col_mean_train)/col_std_train

    train_set, val_set = split_train_val(train_set)

    return train_set, val_set, test_set
```

```python
def update(learning_rate, w, b, lbda, x, y, batch_size):
    y_pred = np.dot(x, w) + b
    idx = y * y_pred
    smaller_idx = np.where(idx < 1)
    bigger_idx = np.where(idx >= 1)
    if len(smaller_idx[0]) == 0:
        w_batch = np.zeros(6)
        b_batch = 0
    else:
        w_batch = -1 * np.dot(x[smaller_idx].T, y[smaller_idx])
        b_batch = np.sum(-1 * y[smaller_idx])

    w  = w - learning_rate * (1/batch_size * w_batch + lbda*w)
    b = b - learning_rate * 1/batch_size * b_batch
    return w, b

def validate(val_set, w, b):
    num_correct = 0
    val_set_x = np.squeeze(val_set[:,:-1])
    val_set_y = val_set[:,-1]
    pred_y = np.dot(val_set_x, w)
    idx = pred_y * val_set_y
    num_correct = len(idx[idx > 0])
    return num_correct/len(val_set)

def predict(w, b, test_set, dir):
    idx = []
    preds = []
    with open(dir + 'labels_test.csv', 'w+') as f:
        writer = csv.writer(f, quoting=csv.QUOTE_NONNUMERIC)
        writer.writerow(["Example","Label"])
        for i in range(len(test_set)):
            y = np.dot(w, test_set[i,:]) + b
            if y >= 0:
                pred = '>50K'
            else:
                pred = '<=50K'
            idx = "'"+ str(i)+"'"
            writer.writerow([idx,pred])
```