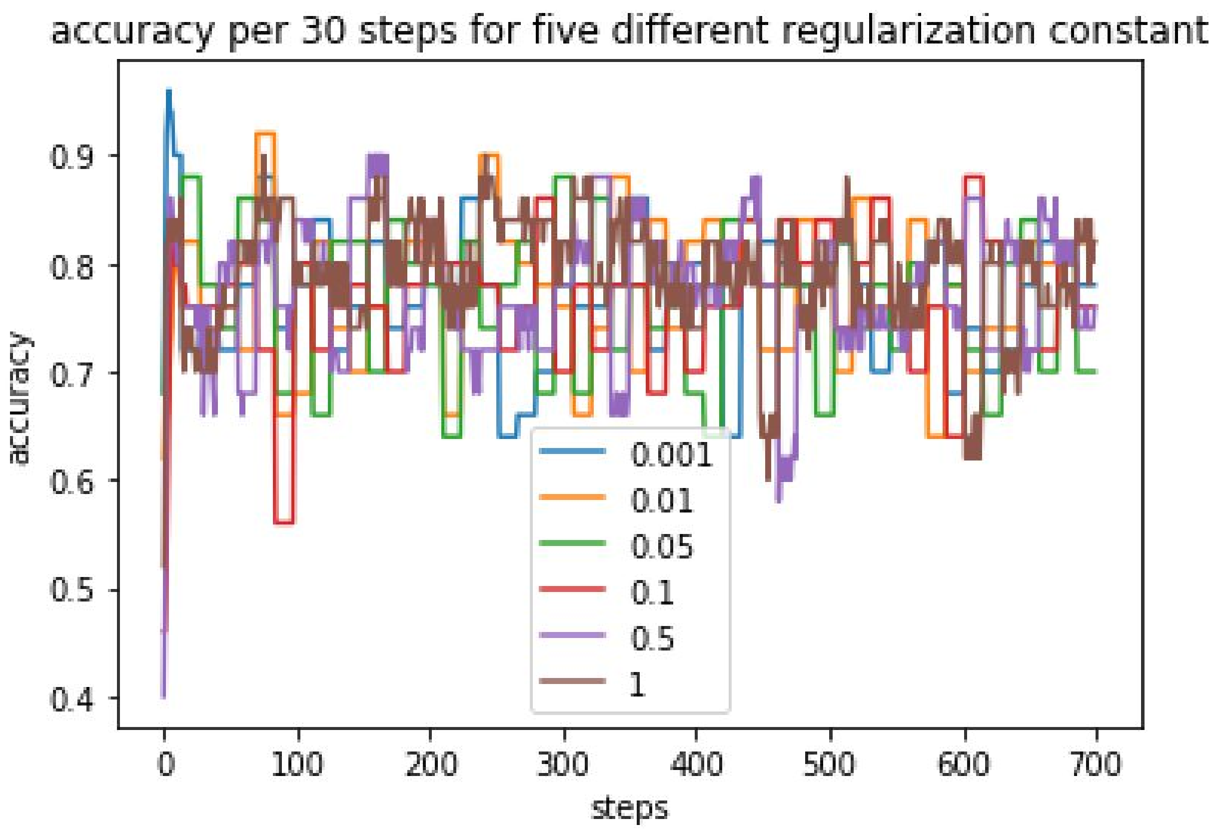
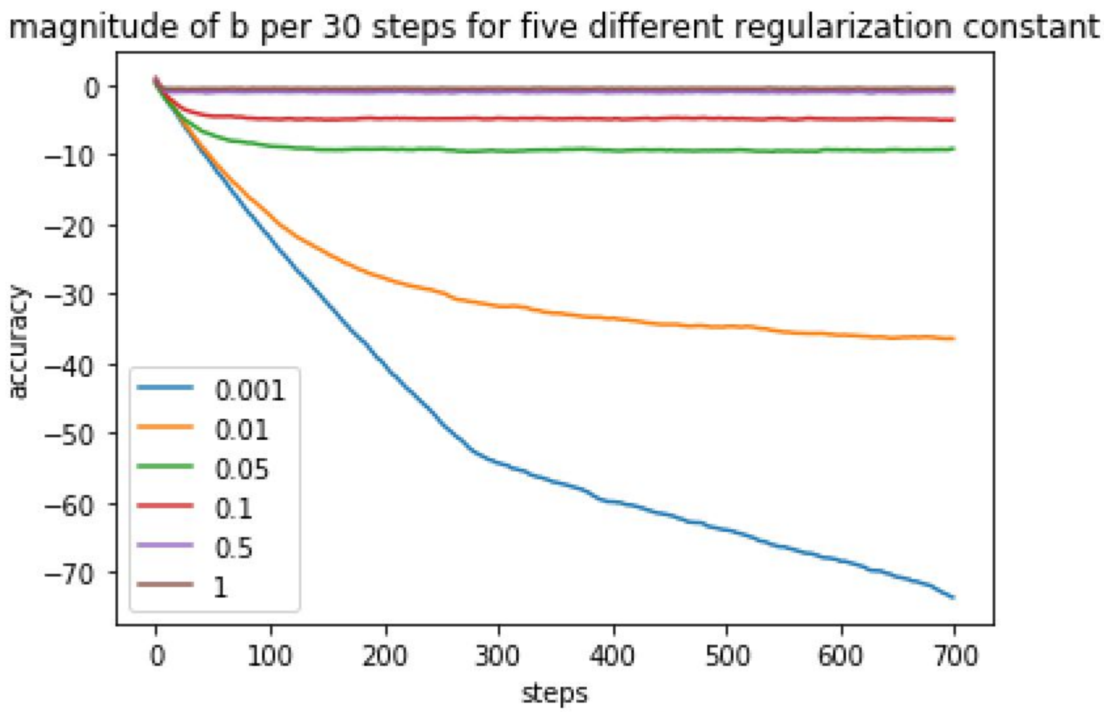
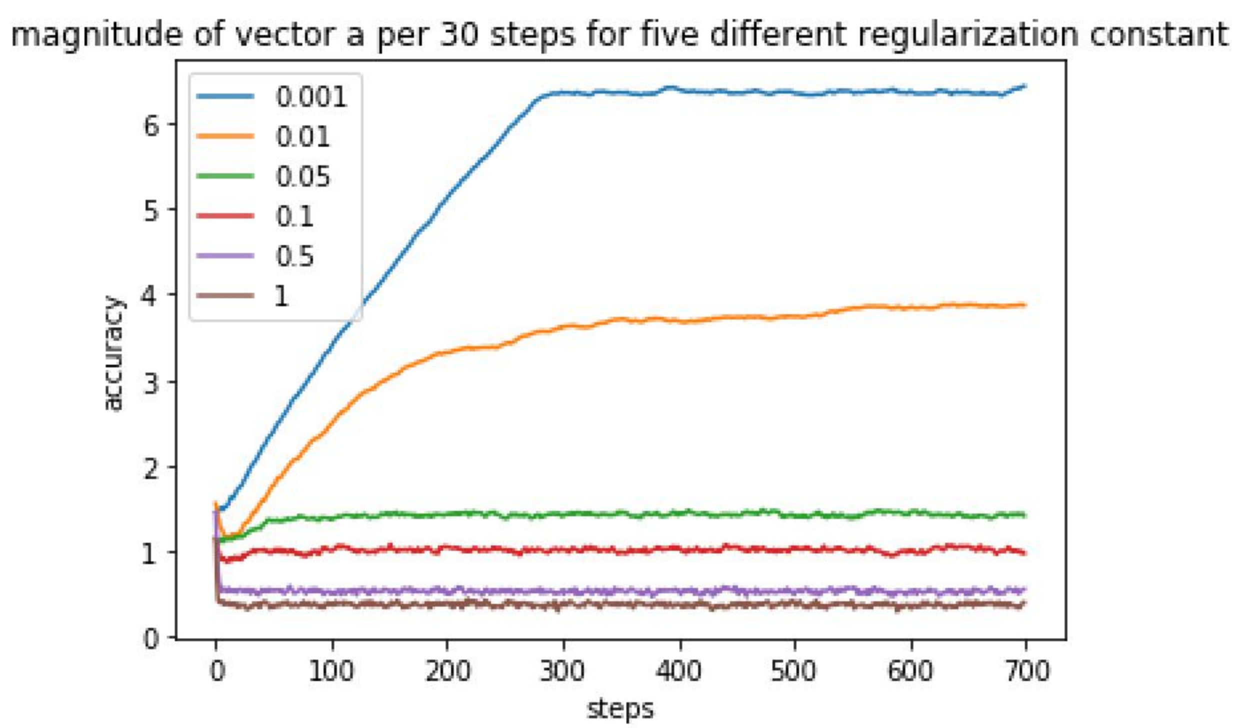
Part 1 : Our Best Accuracy

Part 2: Plot of the validation accuracy every 30 steps



Part 3 : Plot of the magnitude of the coefficient vector every 30 steps





Part 4:

Out best value of regularization constant is 1, because after cross validation, this regularization constant can tune the classifier to have the best accuracy among other choices. We choose initial learning rate as 0.02 at season via formula: $n/m + 0.01 * season$ with $n = 50,\ m = 1$. For each season, update and decrease learning rate by increase the index of season. The reason to choose this learning rate is to prevent the learning rate too high or too low, which would lead to overshoot or slowly reach to best result.

Part 5: A screenshot of your code.

- *Training of an SVM, including but not limited to SGD*

```
lambda_acc = []

step_acc_total = []
a_plot_total = []
b_plot_total = []
for each_lambda_value in lambda_value:
    # Split Train and Validation
    np.random.shuffle(train_combine)
    validation_X = train_combine[39561:, :6]
    validation_Y = train_combine[39561:, -1]

    train_X_Y = train_combine[:39561]

    #intialize a and b
    a = np.zeros(6)

    for i in range(6):
        a[i] = random.randint(1,100)/100
    b = random.randint(1,100)/100
    a = a.reshape((1,6))
    a_plot = []
    b_plot = []
    step_acc = []
    for s in range(1, season_num + 1):
        print("season: ", s)
        learning_rate = m / (0.01 * s + n)

        np.random.shuffle(train_X_Y)
        held_out_X = train_X_Y[:50, :6]
        held_out_Y = train_X_Y[:50, -1]

        season_train_X_Y = train_X_Y[50:]

        for step in range(step_num):
            np.random.shuffle(season_train_X_Y)
            batch_train_X = season_train_X_Y[:batch_num, :6]
            batch_train_Y = season_train_X_Y[:batch_num, -1]
            gradient_a = np.zeros(6)
            gradient_a = gradient_a.reshape((1,6))
            gradient_b = 0
            for batch in range(batch_num):
                if batch_train_Y[batch] * np.dot(a,(batch_train_X[batch].T)) >= 1 :
                    gradient_a += 0
                    gradient_b += 0
                else :
                    gradient_a += -batch_train_Y[batch] * (batch_train_X[batch])
                    gradient_b += -batch_train_Y[batch]
            gradient_a = gradient_a * (-1/batch_num) - each_lambda_value * a
            gradient_b = gradient_b * (-1/batch_num) - each_lambda_value * b

            # update a and b
            a = a + learning_rate * gradient_a
            b = b + learning_rate * gradient_b

            correct = 0
            wrong = 0
            #for each 30 steps evaluation
            if step % 30 == 0 :
                for held_index in range(len(held_out_X)) :
                    if np.sign(np.dot(a, held_out_X[held_index].T) + b) == np.sign(held_out_Y[held_index]) :
                        correct += 1
                    else:
                        wrong += 1
                step_acc.append(correct / (correct + wrong))
                a_plot.append(np.linalg.norm(a))
                b_plot.append(b)
```

The rest of code refer to page 6+

- *Choose the best regularization constant:*

```
    correct_val = 0
    wrong_val = 0
    # acc for each regulization constant
    for validation_index in range(len(validation_X)) :
        if np.sign(np.dot(a, validation_X[validation_index].T) + b) ==  np.sign(validation_Y[validation_index]) :
            correct_val += 1
        else :
            wrong_val += 1
    print("lambda", correct_val / (correct_val + wrong_val))

    lambda_acc.append(correct_val / (correct_val + wrong_val))

lambda_best = lambda_value[np.argmax(lambda_acc)]
```

- *Testing of an SVM.*

```
# Test
# Initialize A and B
a = np.zeros(6)

for i in range(6):
    a[i] = random.randint(1,100)/100
b = random.randint(1,100)/100
a = a.reshape((1,6))

for s in range(1, 1 + season_num):
    learning_rate = m / (0.01 * s + n)
    print(s)
    for step in range(step_num):
        np.random.shuffle(train_combine)
        batch_train_X = train_combine[:batch_num, :6]
        batch_train_Y = train_combine[:batch_num, -1]
        gradient_a = np.zeros(6)
        gradient_a = gradient_a.reshape((1,6))
        gradient_b = 0
        for batch in range(batch_num):
            if batch_train_Y[batch] * np.dot(a,(batch_train_X[batch].T)) >= 1 :
                gradient_a += 0
                gradient_b += 0
            else :
                gradient_a += -batch_train_Y[batch] * (batch_train_X[batch])
                gradient_b += -batch_train_Y[batch]
        gradient_a = gradient_a * (-1/batch_num) - lambda_best * a
        gradient_b = gradient_b * (-1/batch_num) - lambda_best * b

        # update a and b
        a = a + learning_rate * gradient_a
        b = b + learning_rate * gradient_b


# Predict with chosen parameters
predict_result = []

for test_index in range(len(test_feature_scaled)) :
    if np.sign(np.dot(a, test_feature_scaled[test_index].T) + b) ==  1.0:
        predict_result.append(1)
    else :
        predict_result.append(-1)
```

Part 6: All Code

```
import numpy as np
import pandas
import random

import matplotlib.pyplot as plt


train_feature = []
train_label = []

test_feature = []
# read in data
with open('train_data.txt', 'r') as train_file:
    for line in train_file:
        current_line = line.strip().split(', ')
        current_feature = list(float(current_line[i]) for i in [0, 2, 4, 10, 11, 12])


        if current_line[-1] == '<=50K':
            train_label.append(-1)
        elif current_line[-1] == '>50K':
            train_label.append(1)

        train_feature.append(current_feature)

with open('test_data.txt', 'r') as test_file:
    for line in test_file:
        current_line = line.strip().split(', ')
        current_feature = list(float(current_line[i]) for i in [0, 2, 4, 10, 11, 12])
        test_feature.append(current_feature)


train_feature = np.asarray(train_feature)
test_feature = np.asarray(test_feature)
train_label = np.asarray(train_label)
train_label = train_label.reshape((43957, 1))
# Normalize Train
train_feature_mean = np.mean(train_feature, axis = 0)
train_feature_std = np.std(train_feature, axis = 0)

train_feature_scaled = (train_feature - train_feature_mean) / train_feature_std

# Normalize Test
test_feature_mean = np.mean(test_feature, axis = 0)
test_feature_std = np.std(test_feature, axis = 0)

test_feature_scaled = (test_feature - test_feature_mean) / test_feature_std
```

```
# Combine Training data and Training Label
train_combine = np.concatenate((train_feature_scaled, train_label), axis=1)

##############################################################################################################

# parameter combination
season_num = 50
step_num = 400
batch_num = 10
m = 1
n = 50

lambda_value = [0.001, 0.01, 0.05, 0.1, 0.5, 1]

#lambda_value = [1]

lambda_acc = []

step_acc_total = []
a_plot_total = []
b_plot_total = []
for each_lambda_value in lambda_value:
    # Split Train and Validation
    np.random.shuffle(train_combine)
    validation_X = train_combine[39561:, :6]
    validation_Y = train_combine[39561:, -1]

    train_X_Y = train_combine[:39561]

    #intialize a and b
    a = np.zeros(6)

    for i in range(6):
        a[i] = random.randint(1,100)/100
    b = random.randint(1,100)/100
    a = a.reshape((1,6))
    a_plot = []
    b_plot = []
    step_acc = []
    for s in range(1, season_num + 1):
        print("season: ", s)
        learning_rate = m / (0.01 * s + n)

        np.random.shuffle(train_X_Y)
        held_out_X = train_X_Y[:50, :6]
        held_out_Y = train_X_Y[:50, -1]
```

```
    season_train_X_Y = train_X_Y[50:]

    for step in range(step_num):
        np.random.shuffle(season_train_X_Y)
        batch_train_X = season_train_X_Y[:batch_num, :6]
        batch_train_Y = season_train_X_Y[:batch_num, -1]
        gradient_a = np.zeros(6)
        gradient_a = gradient_a.reshape((1,6))
        gradient_b = 0
        for batch in range(batch_num):
            if batch_train_Y[batch] * np.dot(a,(batch_train_X[batch].T)) >= 1 :
                gradient_a += 0
                gradient_b += 0
            else :
                gradient_a += -batch_train_Y[batch] * (batch_train_X[batch])
                gradient_b += -batch_train_Y[batch]
        gradient_a = gradient_a * (-1/batch_num) - each_lambda_value * a
        gradient_b = gradient_b * (-1/batch_num) - each_lambda_value * b

        # update a and b
        a = a + learning_rate * gradient_a
        b = b + learning_rate * gradient_b

        correct = 0
        wrong = 0
        #for each 30 steps evaluation
        if step % 30 == 0 :
            for held_index in range(len(held_out_X)) :
                if np.sign(np.dot(a, held_out_X[held_index].T) + b) == np.sign(held_out_Y[held_index]) :
                    correct += 1
                else:
                    wrong += 1
            step_acc.append(correct / (correct + wrong))
            a_plot.append(np.linalg.norm(a))
            b_plot.append(b)

a_plot_total.append(a_plot)
b_plot_total.append(b_plot)
step_acc_total.append(step_acc)

correct_val = 0
wrong_val = 0
# acc for each regulization constant
for validation_index in range(len(validation_X)) :
    if np.sign(np.dot(a, validation_X[validation_index].T) + b) ==  np.sign(validation_Y[validation_index]) :
        correct_val += 1
```

```
        else :
            wrong_val += 1
    print("lambda", correct_val / (correct_val + wrong_val))

    lambda_acc.append(correct_val / (correct_val + wrong_val))

lambda_best = lambda_value[np.argmax(lambda_acc)]




##################################################################################

# Test
# Initialize A and B
a = np.zeros(6)

for i in range(6):
    a[i] = random.randint(1,100)/100
b = random.randint(1,100)/100
a = a.reshape((1,6))

for s in range(1, 1 + season_num):
    learning_rate = m / (0.01 * s + n)
    print(s)
    for step in range(step_num):
        np.random.shuffle(train_combine)
        batch_train_X = train_combine[:batch_num, :6]
        batch_train_Y = train_combine[:batch_num, -1]
        gradient_a = np.zeros(6)
        gradient_a = gradient_a.reshape((1,6))
        gradient_b = 0
        for batch in range(batch_num):
            if batch_train_Y[batch] * np.dot(a,(batch_train_X[batch].T)) >= 1 :
                gradient_a += 0
                gradient_b += 0
            else :
                gradient_a += -batch_train_Y[batch] * (batch_train_X[batch])
                gradient_b += -batch_train_Y[batch]
        gradient_a = gradient_a * (-1/batch_num) - lambda_best * a
        gradient_b = gradient_b * (-1/batch_num) - lambda_best * b

        # update a and b
        a = a + learning_rate * gradient_a
        b = b + learning_rate * gradient_b
```

```
# Predict with chosen parameters
predict_result = []

for test_index in range(len(test_feature_scaled)) :
    if np.sign(np.dot(a, test_feature_scaled[test_index].T) + b) ==  1.0:
        predict_result.append(1)
    else :
        predict_result.append(-1)

# Output to submission file
with open('submission.txt', 'w') as predict_file:
    for result in predict_result:
        if result == 1:
            predict_file.write('>50K\n')
        else:
            predict_file.write('<=50K\n')
```