CS498 AML,AMO HW2

Ankush Singhal, Shrashti Singhal

TOTAL POINTS

100 / 100

QUESTION 1

1 Screenshot of best accuracy 15 / 15

+ **10.5 pts** Accuracy in [70.0, 76.0) if they show SVM update equations in code snippet
+ **15 pts** Accuracy in range [78.4, 100]
+ **12 pts** Accuracy in [76.0, 78.4) {Since the test accuracy is 75.9 with all positive labels}
+ **6 pts** Other accuracy values if they show SVM update equations in code snippet
- **15 pts** Otherwise

+ 15 Point adjustment

QUESTION 2

2 Plot of accuracy for different regularization constants 20 / 20

- **5 pts** One plot is missing.
- **5 pts** One plot is not correct, e.g., curves not converged etc.
- **2 pts** If one doesn't plot the curves as required, i.e, [1,1e-1,1e-2,1e-3], but plots enough curves.
+ **20 pts** Full points.

+ 20 Point adjustment

QUESTION 3

3 Plot of magnitude of the coeff for different regularization constants 20 / 20

- **5 pts** One plot is missing.
- **5 pts** One plot is not correct, e.g., curves not converged etc.
- **2 pts** If one doesn't plot the curves as required, i.e, [1,1e-1,1e-2,1e-3], but plots enough number of curves.
- **0 pts** Correct

+ 20 Point adjustment

QUESTION 4

4 Best Estimate of reg constant and learning rate + explanation 25 / 25

- **0 pts** Correct
- **0.5 pts** If one only says trying several lrs (without any detail lr values) but the one mentioned in the text book is the best.
- **1 pts** If one just simply says that lr = 1/(0.01*season +50) seems to give a better accuracy without anymore analysis and explanation.
- **12.5 pts** Analysis for the learning rate is missing.
- **12.5 pts** Analysis for lambda is missing.

+ 25 Point adjustment

QUESTION 5

5 Screenshot of Code 20 / 20

- **0 pts** Correct
- **5 pts** SGD is incorrect

+ 20 Point adjustment

QUESTION 6

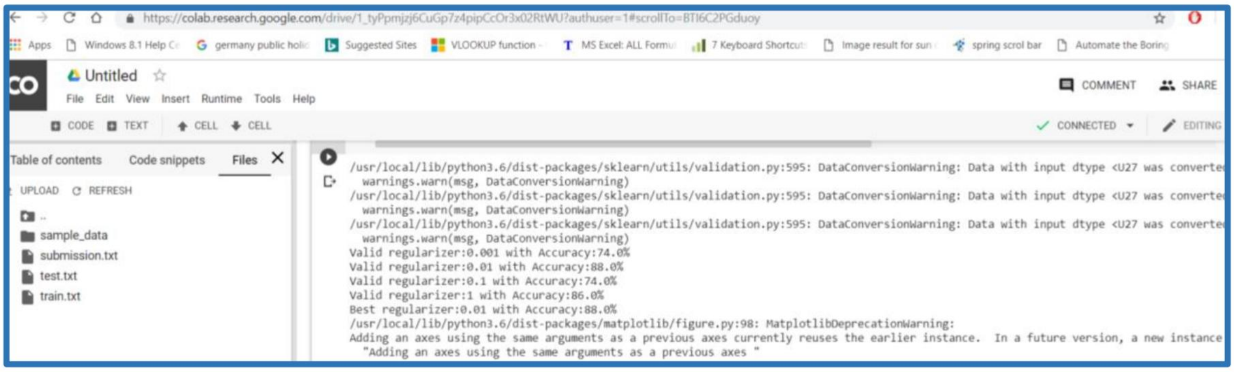
6 Late 0 / 0

- **5 pts** 1 day
- **10 pts** 2 days
- **15 pts** 3 days
- **20 pts** 4 days
- **25 pts** 5 days
- **30 pts** 6 days

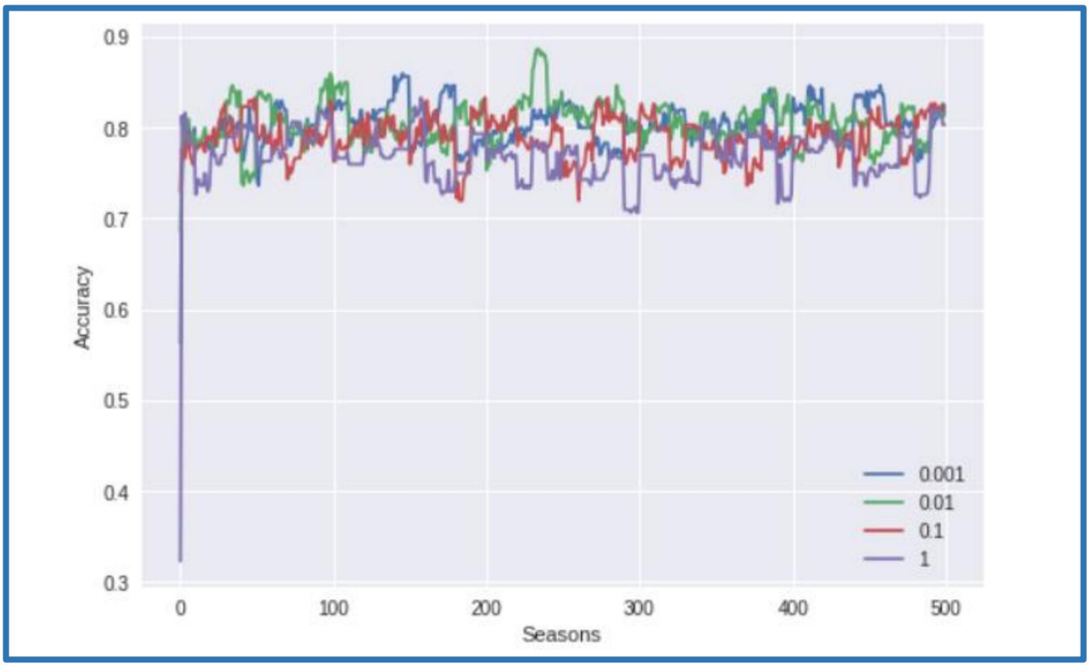
✓ **- 0 pts On time.**

By: Ankush Singhal & Shrashti Singhal

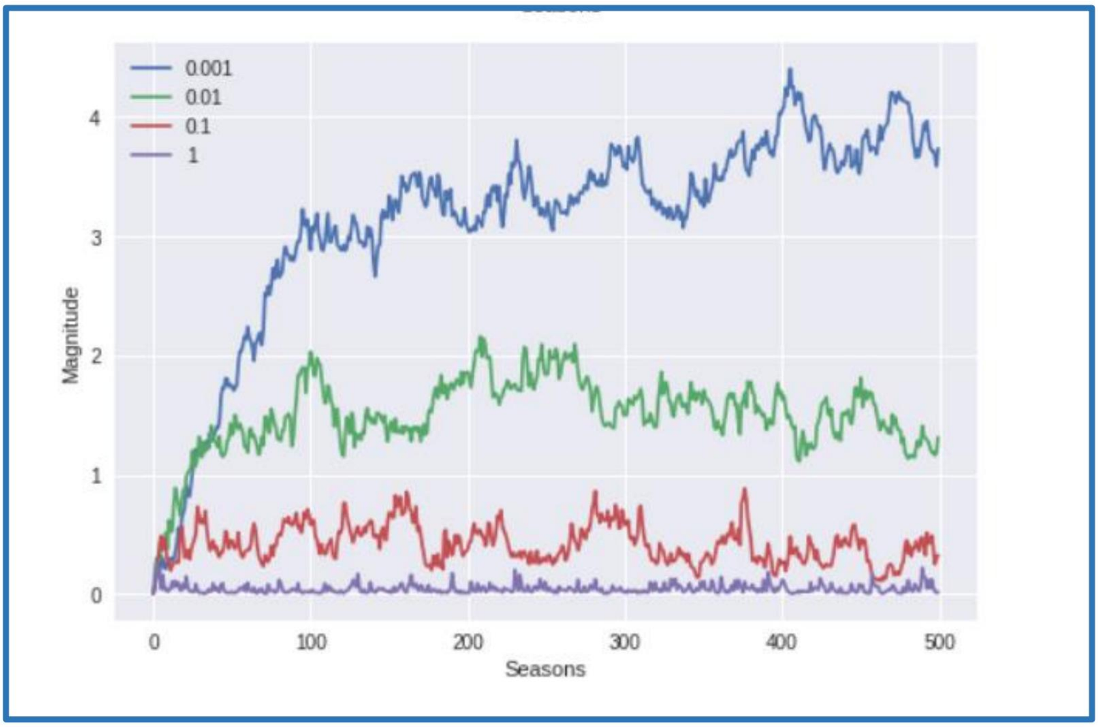
Screenshot of Best Accuracy retrieved from Auto Grader on Gradescope



Plot of Validation Accuracy



Plot of the magnitude of the coefficient vector



Estimate of Best value of the regularization Constant

We have tried different regularization constants. The best estimate of regularization constant is 0.01, which yields our best accuracy of 81.99%

Lambda	Accuracy
0.000001	79.86%
0.00001	79.00%
0.0001	80.49%
0.001	80.59%
0.01	**81.99%**
0.1	80.99%
1	81.35%

Why this regularization Constant is a good Value

Since we achieved the lowest cross validation errors (high accuracy, which is desirable) with 0.01 regularization constant, therefore 0.01 regularization constant is a good value.

Choice for the Learning Rate

Learning Rate selected: (1/(0.01*s+50)

Reasons for selecting this Learning Rate.

In order to determine the appropriate rate of moment towards optimal weight, learning rate selected is (1/(0.01*s+50)

The learning rate has to be bigger at start and slower towards the end so that the values a and b gets refined correctly. The learning rate should not be very high or low at the start. If it's too low, it will take more steps to reach to the right value of a and b. If it's too high, it might not lead in proper direction.

The selected learning rate (1/(0.01*s+50) gives us the right balance.

CODE:

```python
import csv
from sklearn import preprocessing
import numpy as np
from matplotlib import pyplot as plt
import random

def loadCsv(filename):
  lines = csv.reader(open(filename,"r"))
  dataset = list(lines)
  return dataset

def stochasticGradientDescent(train_input_x, train_input_y, regularizer, train_sample_amount):

  ## INITIALIZE LIST OF ACCURACY ANF MAGNITUDE
  list_accuracy = []
  list_magnitude = []
  ## INITIALIZE a AND b
  a = np.array([0, 0, 0, 0, 0, 0])
  b = 0.00

  # GRADIENT DESCENT
  ## TRAIN
  amount_epoch = 50
  amount_step = 300
  amount_validation = 50

  for iter_epoch in range(amount_epoch):
    # step_length
    step_length = 1.0 / ((0.01 * iter_epoch) + 50)
    # data of the whole epoch
    index_epoch = np.random.choice(train_sample_amount, size=amount_step + amount_validation, replace=False)
    train_input_x_epoch = train_input_x[index_epoch, :]
    train_input_y_epoch = train_input_y[index_epoch]

    # training data
    index_step = np.random.choice(train_input_x_epoch.shape[0], size=amount_step, replace=False)
    train_input_x_step = train_input_x_epoch[index_step, :]
    train_input_y_step = train_input_y_epoch[index_step]

    #validation data
    train_input_x_validation = np.delete(train_input_x_epoch, index_step, axis=0)
    train_input_y_validation = np.delete(train_input_y_epoch, index_step, axis=0)

    # renew a and b
    for iter_step in range(amount_step):
      xi = train_input_x_step[iter_step, :]
      yi = train_input_y_step[iter_step]
      gi = yi * ((a).dot(xi) + b)

      if (gi >= 1):
        a = a - step_length * regularizer * a
      else:
        a = a - step_length * (regularizer * a - yi * xi)
        b = b + step_length * yi

      # EVERY 30 STEPS
      if(iter_step % 30 == 0):
        # predict label of training set and get accuracy
        correct_amount = 0
        for iter_y in range(amount_step):
          if train_input_y_step[iter_y] * ((a).dot(train_input_x_step[iter_y, :]) + b) > 0:
            correct_amount = correct_amount + 1
        accuracy = float(correct_amount / amount_step)
        list_accuracy.append(accuracy)

        # get magnitude
        magnitude = (a).dot(a.T)
        list_magnitude.append(magnitude)
  correct_amount = predict(amount_validation, train_input_x_validation, train_input_y_validation,a,b)
  accuracy_validation = correct_amount / amount_validation
  return a, b, accuracy_validation, list_accuracy, list_magnitude

def predict(amount_validation, train_input_x_validation, train_input_y_validation,a,b):
  # predict label of validation set
  correct_amount = 0
  for iter_y in range(amount_validation):
    if train_input_y_validation[iter_y] * ((a).dot(train_input_x_validation[iter_y, :]) + b) > 0:
      correct_amount = correct_amount + 1
  return correct_amount
```

```python
# output result in csv file
def writeCsvFile(filename, test_output_y):
  with open(filename, "w") as test_output_file:
    test_output_writer = csv.writer(test_output_file)
    # write content
    test_sample_amount = test_output_y.shape[0]
    content = []
    for iter in range(test_sample_amount):
      string_index = "'" + str(iter) + "'"
      content.append([test_output_y[iter]])
    test_output_writer.writerows(content)

def splitDataset(train_input_x, train_input_y, splitRatio):
  train_size = int(len(train_input_x) * splitRatio)
  train_set_X = []
  train_set_Y = []
  copyX = list(train_input_x)
  copyY = list(train_input_y)
  while (len(train_set_X) < train_size):
    index = random.randrange(len(copyX))
    train_set_X.append(copyX.pop(index))
    train_set_Y.append(copyY.pop(index))
  return [train_set_X, train_set_Y, copyX, copyY]

def main():

  # READ IN TRAINING DATA
  train_input_file = 'train.txt'
  # Laod Train data
  train_input_data_list = loadCsv(train_input_file)

    # change input data from list to array
  train_input_data = np.array(train_input_data_list)

    # get training data set size
  train_sample_amount = train_input_data.shape[0]
  train_feature_amount = train_input_data.shape[1]

  index_x = [0, 2, 4, 10, 11, 12]
  feature_amount = 6
  splitRatio = 0.90
```

```python
    # extract data of feature and label
  train_input_x = train_input_data[:, index_x]
  train_input_x = np.array(train_input_x)

  # classify training labels
  train_input_y = train_input_data[:, train_feature_amount-1]
  for iter_y in range(0, train_sample_amount):
    if train_input_y[iter_y] == ' <=50K':   # <=50K
      train_input_y[iter_y] = -1
    else:                                   # >50K
      train_input_y[iter_y] = 1
  train_input_y = np.array(train_input_y).astype(int)

  #Segregating dataset into train set and valiadtion set
  train_input_x, train_input_y, validation_train_input_x, validation_train_input_y = splitDataset(train_input_x, train_input_y, splitRatio)

  train_input_x = np.array(train_input_x)
  train_input_y = np.array(train_input_y).astype(int)
  validation_train_input_x = np.array(validation_train_input_x)
  validation_train_input_y = np.array(validation_train_input_y).astype(int)

  train_sample_amount = train_input_x.shape[0]
  validation_train_sample_amount = validation_train_input_x.shape[0]

    # READ IN TESTING DATA
  test_input_file = 'test.txt'
  # Laod Train data
  test_input_data_list = loadCsv(test_input_file)

  # change input data from list to array
  test_input_data = np.array(test_input_data_list)
  test_sample_amount = test_input_data.shape[0]

  # extract data of feature and label
  test_input_x = test_input_data[:, index_x]
  test_input_x = np.array(test_input_x)
```

```python
  # RESCALE - Scale these variables so that each has unit variance and subtract the mean so that each has zero mean
  train_input_x_rescaled = preprocessing.scale(train_input_x, axis=0, with_mean=True, with_std=True)
  np.array(train_input_x_rescaled).astype(float)
  validation_train_input_x_rescaled = preprocessing.scale(validation_train_input_x, axis=0, with_mean=True, with_std=True)
  np.array(validation_train_input_x_rescaled).astype(float)
  test_input_x_rescaled = preprocessing.scale(test_input_x, axis=0, with_mean=True, with_std=True)
  np.array(test_input_x_rescaled).astype(float)

  ## Train the Train Set
  train_regularizer = [0.001, 0.01, 0.1, 1]
  accuray = []
  magnitude = []
  for regularizer in train_regularizer:
    [current_a, current_b, current_accuracy, list_accuracy, list_magnitude] = stochasticGradientDescent(train_input_x_rescaled, train_input_y
    accuray.append(list_accuracy)
    magnitude.append(list_magnitude)

  #Calculate Best regularizer for 10% validation dataset
  best_accuracy = 0
  best_a = np.array([0, 0, 0, 0, 0, 0])
  best_b = 0.00
  best_regularizer = 0


  for regularizer in train_regularizer:
    [valid_current_a, valid_current_b, valid_current_accuracy, valid_list_accuracy, valid_list_magnitude] = stochasticGradientDescent(validat
    print('Valid regularizer:{} with Accuracy:{}%'.format(regularizer, (valid_current_accuracy*100)))
    if valid_current_accuracy > best_accuracy:
      best_a = valid_current_a
      best_b = valid_current_b
      best_accuracy = valid_current_accuracy
      best_regularizer = regularizer
  print('Best regularizer:{} with Accuracy:{}%'.format(best_regularizer, (best_accuracy*100)))

```

```python
  ## TEST
  test_output_y = []
  for iter_y in range(test_sample_amount):
    if best_a.dot(test_input_x_rescaled[iter_y, :]) + best_b > 0:
      test_output_y.append('>50K')
    else:
      test_output_y.append('<=50K')
  test_output_y = np.array(test_output_y)
  writeCsvFile("submission.txt", test_output_y)

  ## DRAW
  image_a = plt.figure()
  for iter in range(len(train_regularizer)):
    y = accuray[iter]
    x = np.arange(len(y))

    image_accuracy = image_a.add_subplot(111)
    image_accuracy.plot(x, y)

  image_accuracy.legend(train_regularizer)
  image_accuracy.set_xlabel('Seasons')
  image_accuracy.set_ylabel('Accuracy')
  image_a.show()

  image_m = plt.figure()
  for iter in range(len(train_regularizer)):
    y = magnitude[iter]
    x = np.arange(len(y))

    image_magnitude = image_m.add_subplot(111)
    image_magnitude.plot(x, y)

  image_magnitude.legend(train_regularizer)
  image_magnitude.set_xlabel('Seasons')
  image_magnitude.set_ylabel('Magnitude')
  image_m.show()


main()
```