

# 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  $lr = 1/(0.01 * \text{season} + 50)$  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

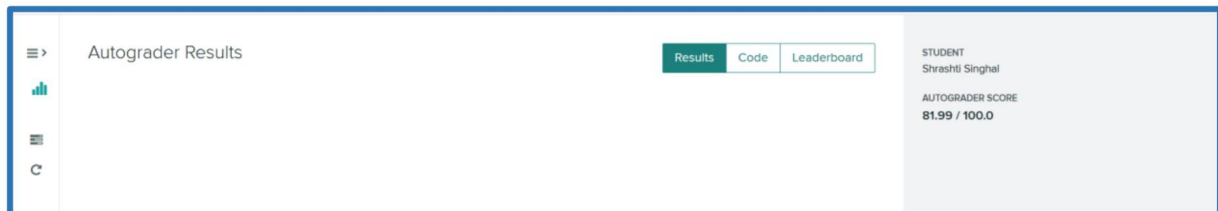
### 6 Late **0 / 0**

- **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.

## APPLIED MACHINE LEARNING Assignment 2

By: Ankush Singhal & Shrashti Singhal

### Screenshot of Best Accuracy retrieved from Auto Grader on Gradescope

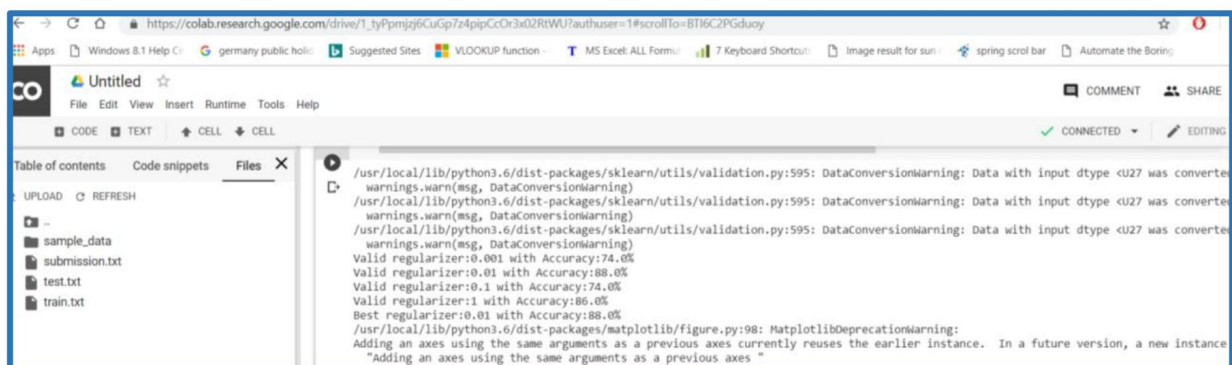


The screenshot shows the 'Autograder Results' page on Gradescope. It features a sidebar with navigation icons, a main area with tabs for 'Results', 'Code', and 'Leaderboard', and a right-hand panel displaying student information and the final score.

STUDENT
Shrashti Singhal

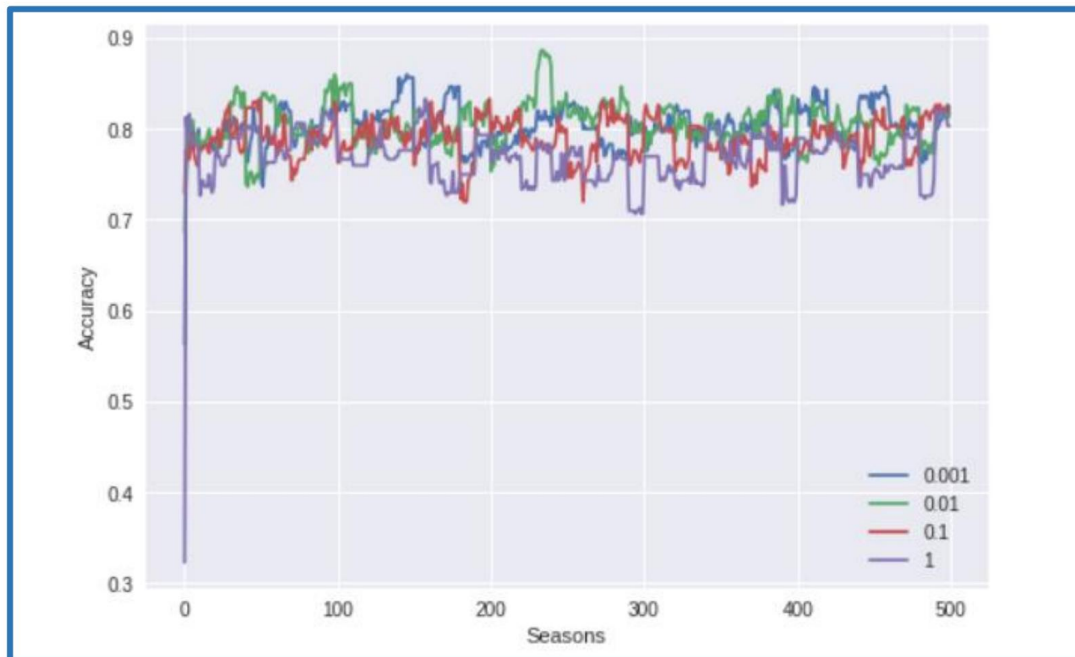
AUTOGRADER SCORE
81.99 / 100.0



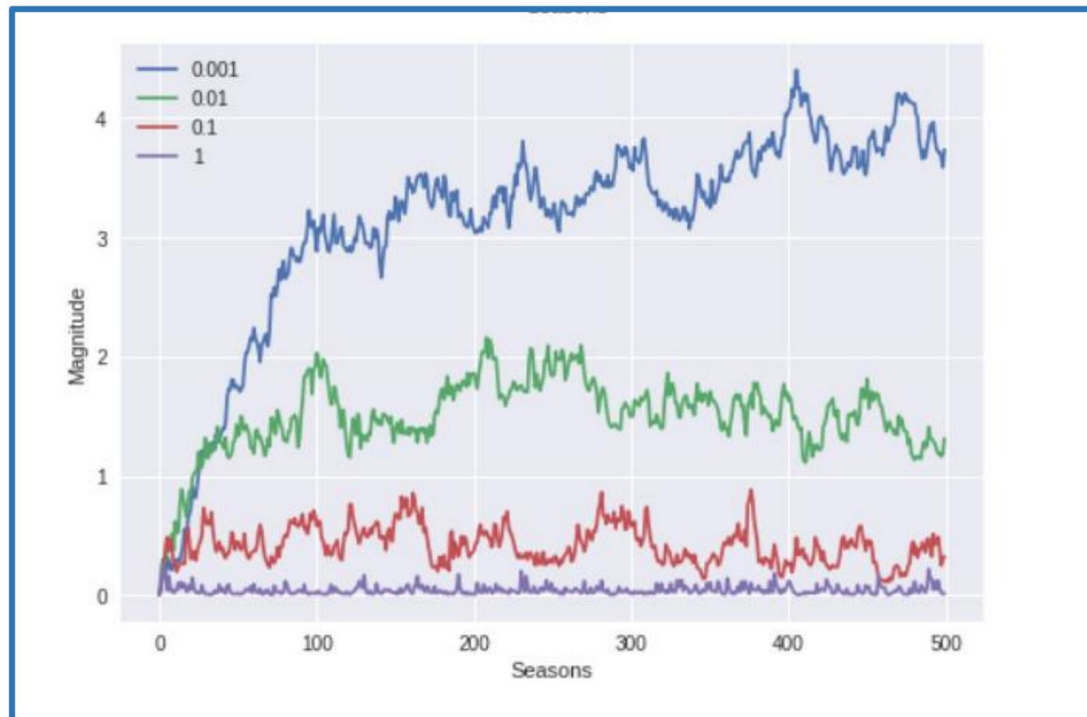
The screenshot shows a Google Colab notebook interface. The left sidebar displays a file explorer with folders like 'sample\_data' and files like 'submission.txt', 'test.txt', and 'train.txt'. The main code cell contains the output of a model evaluation, showing accuracy for different regularizers.

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype <U27 was converted to dtype <U27. This will likely fail.
warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype <U27 was converted to dtype <U27. This will likely fail.
warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype <U27 was converted to dtype <U27. This will likely fail.
warnings.warn(msg, DataConversionWarning)
Valid regularizer:0.001 with Accuracy:74.0%
Valid regularizer:0.01 with Accuracy:88.0%
Valid regularizer:0.1 with Accuracy:74.0%
Valid regularizer:1 with Accuracy:86.0%
Best regularizer:0.01 with Accuracy:88.0%
/usr/local/lib/python3.6/dist-packages/matplotlib/figure.py:98: MatplotlibDeprecationWarning:
Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will be created.
"Adding an axes using the same arguments as a previous axes "
```

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%

<b>Lambda</b>	<b>Accuracy</b>
0.000001	79.86%
0.00001	79.00%
0.0001	80.49%
0.001	80.59%
<b>0.01</b>	<b>81.99%</b>
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:

```
1 import csv
2 from sklearn import preprocessing
3 import numpy as np
4 from matplotlib import pyplot as plt
5 import random
6
7 def loadCsv(filename):
8     lines = csv.reader(open(filename,"r"))
9     dataset = list(lines)
10    return dataset
11
12 def stochasticGradientDescent(train_input_x, train_input_y, regularizer, train_sample_amount):
13
14    ## INITIALIZE LIST OF ACCURACY ANF MAGNITUDE
15    list_accuracy = []
16    list_magnitude = []
17    ## INITIALIZE a AND b
18    a = np.array([0, 0, 0, 0, 0, 0])
19    b = 0.00
20
21    # GRADIENT DESCENT
22    ## TRAIN
23    amount_epoch = 50
24    amount_step = 300
25    amount_validation = 50
26
27    for iter_epoch in range(amount_epoch):
28        # step_length
29        step_length = 1.0 / ((0.01 * iter_epoch) + 50)
30        # data of the whole epoch
31        index_epoch = np.random.choice(train_sample_amount, size=amount_step + amount_validation, replace=False)
32        train_input_x_epoch = train_input_x[index_epoch, :]
33        train_input_y_epoch = train_input_y[index_epoch]
34
35        # training data
36        index_step = np.random.choice(train_input_x_epoch.shape[0], size=amount_step, replace=False)
37        train_input_x_step = train_input_x_epoch[index_step, :]
38        train_input_y_step = train_input_y_epoch[index_step]
39
40    #validation data
41    train_input_x_validation = np.delete(train_input_x_epoch, index_step, axis=0)
42    train_input_y_validation = np.delete(train_input_y_epoch, index_step, axis=0)
43
44    # renew a and b
45    for iter_step in range(amount_step):
46        xi = train_input_x_step[iter_step, :]
47        yi = train_input_y_step[iter_step]
48        gi = yi * ((a).dot(xi) + b)
49
50        if (gi >= 1):
51            a = a - step_length * regularizer * a
52        else:
53            a = a - step_length * (regularizer * a - yi * xi)
54            b = b + step_length * yi
55
56        # EVERY 30 STEPS
57        if(iter_step % 30 == 0):
58            # predict label of training set and get accuracy
59            correct_amount = 0
60            for iter_y in range(amount_step):
61                if train_input_y_step[iter_y] * ((a).dot(train_input_x_step[iter_y, :]) + b) > 0:
62                    correct_amount = correct_amount + 1
63            accuracy = float(correct_amount / amount_step)
64            list_accuracy.append(accuracy)
65
66            # get magnitude
67            magnitude = (a).dot(a.T)
68            list_magnitude.append(magnitude)
69    correct_amount = predict(amount_validation, train_input_x_validation, train_input_y_validation,a,b)
70    accuracy_validation = correct_amount / amount_validation
71    return a, b, accuracy_validation, list_accuracy, list_magnitude
72
73 def predict(amount_validation, train_input_x_validation, train_input_y_validation,a,b):
74    # predict label of validation set
75    correct_amount = 0
76    for iter_y in range(amount_validation):
77        if train_input_y_validation[iter_y] * ((a).dot(train_input_x_validation[iter_y, :]) + b) > 0:
78            correct_amount = correct_amount + 1
79    return correct_amount
80
```



```
81 # output result in csv file
82 def writeCsvFile(filename, test_output_y):
83     with open(filename, "w") as test_output_file:
84         test_output_writer = csv.writer(test_output_file)
85         # write content
86         test_sample_amount = test_output_y.shape[0]
87         content = []
88         for iter in range(test_sample_amount):
89             string_index = "" + str(iter) + ""
90             content.append([test_output_y[iter]])
91             test_output_writer.writerow(content)
92
93 def splitDataset(train_input_x, train_input_y, splitRatio):
94     train_size = int(len(train_input_x) * splitRatio)
95     train_set_X = []
96     train_set_Y = []
97     copyX = list(train_input_x)
98     copyY = list(train_input_y)
99     while (len(train_set_X) < train_size):
100         index = random.randrange(len(copyX))
101         train_set_X.append(copyX.pop(index))
102         train_set_Y.append(copyY.pop(index))
103     return [train_set_X, train_set_Y, copyX, copyY]
104
105 def main():
106
107     # READ IN TRAINING DATA
108     train_input_file = 'train.txt'
109     # Load Train data
110     train_input_data_list = loadCsv(train_input_file)
111
112     # change input data from list to array
113     train_input_data = np.array(train_input_data_list)
114
115     # get training data set size
116     train_sample_amount = train_input_data.shape[0]
117     train_feature_amount = train_input_data.shape[1]
118
119     index_x = [0, 2, 4, 10, 11, 12]
120     feature_amount = 6
121     splitRatio = 0.90
122
123     # extract data of feature and label
124     train_input_x = train_input_data[:, index_x]
125     train_input_y = np.array(train_input_data[:, -1])
126
127     # classify training labels
128     train_input_y = train_input_data[:, train_feature_amount-1]
129     for iter_y in range(0, train_sample_amount):
130         if train_input_y[iter_y] == ' <=50K': # <=50K
131             train_input_y[iter_y] = -1
132         else: # >50K
133             train_input_y[iter_y] = 1
134     train_input_y = np.array(train_input_y).astype(int)
135
136     #Segregating dataset into train set and validation set
137     train_input_x, train_input_y, validation_train_input_x, validation_train_input_y = splitDataset(train_input_x, train_input_y, splitRatio)
138
139     train_input_x = np.array(train_input_x)
140     train_input_y = np.array(train_input_y).astype(int)
141     validation_train_input_x = np.array(validation_train_input_x)
142     validation_train_input_y = np.array(validation_train_input_y).astype(int)
143
144     train_sample_amount = train_input_x.shape[0]
145     validation_train_sample_amount = validation_train_input_x.shape[0]
146
147     # READ IN TESTING DATA
148     test_input_file = 'test.txt'
149     # Load Train data
150     test_input_data_list = loadCsv(test_input_file)
151
152     # change input data from list to array
153     test_input_data = np.array(test_input_data_list)
154     test_sample_amount = test_input_data.shape[0]
155
156     # extract data of feature and label
157     test_input_x = test_input_data[:, index_x]
158     test_input_y = np.array(test_input_data[:, -1])
159
```

```
160 # RESCALE - Scale these variables so that each has unit variance and subtract the mean so that each has zero mean
161 train_input_x_rescaled = preprocessing.scale(train_input_x, axis=0, with_mean=True, with_std=True)
162 np.array(train_input_x_rescaled).astype(float)
163 validation_train_input_x_rescaled = preprocessing.scale(validation_train_input_x, axis=0, with_mean=True, with_std=True)
164 np.array(validation_train_input_x_rescaled).astype(float)
165 test_input_x_rescaled = preprocessing.scale(test_input_x, axis=0, with_mean=True, with_std=True)
166 np.array(test_input_x_rescaled).astype(float)
167
168 ## Train the Train Set
169 train_regularizer = [0.001, 0.01, 0.1, 1]
170 accuracy = []
171 magnitude = []
172 for regularizer in train_regularizer:
173     [current_a, current_b, current_accuracy, list_accuracy, list_magnitude] = stochasticGradientDescent(train_input_x_rescaled, train_input_y
174     accuracy.append(list_accuracy)
175     magnitude.append(list_magnitude)
176
177 #Calculate Best regularizer for 10% validation dataset
178 best_accuracy = 0
179 best_a = np.array([0, 0, 0, 0, 0])
180 best_b = 0.00
181 best_regularizer = 0
182
183
184 for regularizer in train_regularizer:
185     [valid_current_a, valid_current_b, valid_current_accuracy, valid_list_accuracy, valid_list_magnitude] = stochasticGradientDescent(validat
186     print('Valid regularizer:{} with Accuracy:{}'.format(regularizer, (valid_current_accuracy*100)))
187     if valid_current_accuracy > best_accuracy:
188         best_a = valid_current_a
189         best_b = valid_current_b
190         best_accuracy = valid_current_accuracy
191         best_regularizer = regularizer
192 print('Best regularizer:{} with Accuracy:{}'.format(best_regularizer, (best_accuracy*100)))
193
```

```
195 ## TEST
196 test_output_y = []
197 for iter_y in range(test_sample_amount):
198     if best_a.dot(test_input_x_rescaled[iter_y, :]) + best_b > 0:
199         test_output_y.append('>50K')
200     else:
201         test_output_y.append('<=50K')
202 test_output_y = np.array(test_output_y)
203 writeCsvFile("submission.txt", test_output_y)
204
205 ## DRAW
206 image_a = plt.figure()
207 for iter in range(len(train_regularizer)):
208     y = accuracy[iter]
209     x = np.arange(len(y))
210
211     image_accuracy = image_a.add_subplot(111)
212     image_accuracy.plot(x, y)
213
214     image_accuracy.legend(train_regularizer)
215     image_accuracy.set_xlabel('Seasons')
216     image_accuracy.set_ylabel('Accuracy')
217     image_a.show()
218
219 image_m = plt.figure()
220 for iter in range(len(train_regularizer)):
221     y = magnitude[iter]
222     x = np.arange(len(y))
223
224     image_magnitude = image_m.add_subplot(111)
225     image_magnitude.plot(x, y)
226
227     image_magnitude.legend(train_regularizer)
228     image_magnitude.set_xlabel('Seasons')
229     image_magnitude.set_ylabel('Magnitude')
230     image_m.show()
231
232
233 main()
```