

Similarity Report for Parth Chopra and Jon Gill

50% Similarity

Matches

Parth Chopra PS4.py

Jon Gill PS4_Jon_Gill.py

354 - 378

1

326 - 350

```

353 figure, ax = plt.subplots(3, 3, sharex='row',
1  sharey='row')
354 delta, STEP_SIZE = 0.01, 0.01
355 final_stress_values = []
356 iter_no = 1
357
358 for i in range(3):
359     for j in range(3):
360         positions =
np.random.rand(len(similarity), 2)
361         for iteration in range(100):
362             positions -=
(compute_gradient(positions, delta) *
STEP_SIZE)
363
final_stress_values.append(StressCalc(position
s, distance))
364
365 pos = 0
366 for sport in positions:

```

```

325 1 delta = 0.01
326 alpha = 0.01
327 final_stress_vals = []
328 iteration = 1
329
330 for i in range(3):
331     for j in range(3):
332         positions =
np.random.rand(len(similarity), 2)
333         for iteration in range(100):
334             positions -=
(compute_gradient(positions, delta) * alpha)
335
final_stress_vals.append(StressCalc(positions,
distance))
336
337 idx = 0
338 for sport in positions:
339     ax[i][j].plot(sport[0], sport[1],
'.')
```

```

367         ax[i][j].plot(sport[0], sport[1],
368         '.')
369         ax[i][j].text(sport[0], sport[1] +
370         0.01, sport_names[pos])
371         pos += 1
372         ax[i][j].set_title(f'Plot of Sports at
373         Final Positions, Iteration #{iter_no}')
374         ax[i][j].set_xlabel('Final x-position')
375         ax[i][j].set_ylabel('Final y-position')
376         iter_no += 1
377     figure.set_size_inches(15, 15)
378     figure.savefig('PS4_Q7_1.png')
379
380

```

```

340         ax[i][j].text(sport[0], sport[1] +
341         0.01, sport_names[idx])
342         idx += 1
343         ax[i][j].set_title(f'Plot of Sports at
344         Final Positions, Iteration #{iteration}')
345         ax[i][j].set_xlabel('Final x-position')
346         ax[i][j].set_ylabel('Final y-position')
347         iter_no += 1
348
349     figure.set_size_inches(15, 15)
350     figure.savefig('PS4_Q7_1.png')
351
352

```

321 - 336

2

296 - 310

```

320 #YOUR CODE HERE
321 2 figure, ax = plt.subplots()
322
323 MDS_distances = np.zeros(distance.shape)
324
325 for i in range(len(positions)):
326     for j in range(len(positions)):
327         MDS_distances[i][j] =
328         EuclideanDistance(positions[i], positions[j])

```

```

295 2 figure, ax = plt.subplots()
296
297 mds_distances = np.zeros(distance.shape)
298
299 for i in range(len(positions)):
300     for j in range(len(positions)):
301         mds_distances[i][j] =
302         EuclideanDistance(positions[i], positions[j])
303

```

```

329
330 plt.scatter(MDS_distances, distance, s=10,
331             c='green')
332 plt.xlabel('Distances obtained by running MDS')
333 plt.ylabel('Reported distances (d = 1-s)')
334 plt.title('MDS Distances Plotted Against
335           Reported Distances')
336 figure.set_size_inches(10, 6)
337 figure.savefig('PS4_Q6.png')
338

```

```

304 ax.scatter(mds_distances, distance, s=10,
305             c='blue')
306 ax.set_xlabel('Distances from running MDS')
307 ax.set_ylabel('Reported distances $(d = 1-s)$')
308 ax.set_title('MDS Distances vs. Reported
309           Distances')
310 figure.set_size_inches(10, 10)
311 figure.savefig('PS4_Q6.png')
312

```

272 - 284

3

249 - 261

```

271 #YOUR CODE HERE
272 pos = 0
273 figure, ax = plt.subplots()
274
275 for sport in positions:
276     plt.plot(sport[0], sport[1], '.')
277     plt.text(sport[0], sport[1] + 0.01,
278             sport_names[pos])
279     pos += 1
280 plt.title('Plot of Sports at Locations of
281           Minimized Stress')
282 plt.xlabel('Final x-position')
283 plt.ylabel('Final y-position')

```

```

248
249 idx = 0
250 figure, ax = plt.subplots()
251
252 for sport in positions:
253     ax.plot(sport[0], sport[1], '.')
254     ax.text(sport[0], sport[1] + 0.01,
255             sport_names[idx])
256     idx += 1
257 ax.set_title('Sports at Locations of Minimized
258           Stress')
259 ax.set_xlabel('Final x-position')
260 ax.set_ylabel('Final y-position')

```

```

284 figure.set_size_inches(10, 6)
285 figure.savefig('PS4_Q5_2.png')
286

```

```

261 figure.set_size_inches(10, 10)
262 figure.savefig('PS4_Q5_2.png')
263

```

119 - 129

4

118 - 127

```

118 #YOUR CODE HERE
119 def convert_similarity_to_distance(similarity):
120     distance = np.zeros(similarity.shape)
121     for i in range(len(similarity)):
122         for j in range(len(similarity[0])):
123             distance[i][j] = 1 - similarity[i]
124             [j]
125     return distance
126
127 distance =
128     convert_similarity_to_distance(similarity)
129
130 distance_1d = np.reshape(distance,
131                             len(distance) * len(distance[0]))
132
133 similarity_1d = np.reshape(similarity,
134                             len(similarity) * len(similarity[0]))
135

```

```

117
118 def similarity_to_distance(similarity):
119     distance = np.zeros(similarity.shape)
120     for i in range(len(similarity)):
121         for j in range(len(similarity[0])):
122             distance[i][j] = 1 - similarity[i]
123             [j]
124     return distance
125
126 distance = similarity_to_distance(similarity)
127 distance_flattened = distance.flatten()
128 similarity_flattened = similarity.flatten()
129
130 figure, ax = plt.subplots()

```

456 - 464

5

416 - 424

```

455
456 ax.plot(steps, stress_values, color='red')

```

```

415
416 ax.plot(stress_vals, color='red')

```

```

457
458 plt.title('Stress Plotted Over Time')
459 plt.xlabel('Step Number')
460 plt.ylabel('Stress Value')
461 plt.legend(['$a = 0.02$', '$a = 0.05$'])
462
463 figure.set_size_inches(10, 6)
464 figure.savefig('PS4_Q8.png')
465
466

```

```

417
418 ax.set_title('Stress Plotted Over Time')
419 ax.set_xlabel('Step Number')
420 ax.set_ylabel('Stress Value')
421 plt.legend(['$a = 0.02$', '$a = 0.05$'])
422
423 figure.set_size_inches(10, 10)
424 figure.savefig('PS4_Q8.png')
425
426

```

227 - 233

6

208 - 214

```

226
227 6 return (StressCalc(plus_delta, distance) -
StressCalc(minus_delta, distance)) / (2 *
delta)
228
229 def compute_gradient(positions, delta):
230     gradient_matrix = np.zeros(positions.shape)
231     for i in range(len(gradient_matrix)):
232         for j in
range(len(gradient_matrix[0])):
233             gradient_matrix[i][j] =
gradient_at_point(positions, i, j, delta)
234
235     return gradient_matrix

```

```

207 6 delta = .01
208 positions = np.random.rand(len(similarity), 2)
209
210 def compute_gradient(positions, delta):
211     gradient_matrix = np.zeros(positions.shape)
212     for i in range(len(gradient_matrix)):
213         for j in
range(len(gradient_matrix[0])):
214             plus_delta = positions.copy()
215             minus_delta = positions.copy()
216             plus_delta[i][j] += delta

```

All Code

Parth Chopra PS4.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # <div style="background-color: #c1f2a5">
5  #
6  #
7  # # PS4
8  #
9  # In this problem set, you will implement multidimensional scaling (MDS) from scratch. You
   may use standard matrix/vector libraries (e.g. numpy) but you must implement two dimensional
   MDS itself on your own and not use an existing software package. MDS attempts to find an
   arrangement of points such that the distances between points match human-judged
   similarities.
10 #
11 # # Instructions
12 #
13 #
14 #
15 # Remember to do your problem set in Python 3. Fill in `#YOUR CODE HERE`.
16 #
17 # Make sure:
18 # - that all plots are scaled in such a way that you can see what is going on (while still
   respecting specific plotting instructions)
19 # - that the general patterns are fairly represented.
20 # - to label all x- and y-axes, and to include a title.
21 #
22 # </div>
23
24 # In[137]:
```

```
25
26
27 import numpy as np
28 import matplotlib.pyplot as plt
29 # to import the data set
30 from scipy.io import loadmat
31
32
33 # ## Import and examine data
34 #
35 # We will be using a data set from Romney, A. K., Brewer, D. D., & Batchelder, W. H. (1993).
    Predicting Clustering from Semantic Structure. Psychological Science, 4(1), 28-34, via
    https://faculty.sites.uci.edu/mdlee/similarity-data/. The data set is saved in
    PS4_dataset.mat, and includes pairwise similarity measures between 21 sports. Make sure that
    the PS notebook and the data set are in the same directory!
36 #
37 # As our first step, we will download and examine the data:
38 #
39
40 # In[138]:
41
42
43 data_set = loadmat('PS4_dataset.mat')
44 similarity = data_set['similarity']
45 sport_names = data_set['sport_names']
46
47
48 # As we can see, our data contains information for 21 different sports as listed below:
49
50 # In[139]:
51
52
53 print(sport_names)
54
```

```
55
56 # We also have a similarity matrix for each sport, which gives us the psychological
    similarity of that sport with all the other sports in the data:
57
58 # In[140]:
59
60
61 #Look at the first similarity matrix, which corresponds to football's similarity with itself
    and all other sports
62 print(similarity[0])
63
64
65 # ## Q1. Visualize similarity [5pts, HELP]
66
67 # Plot the "similarity" measures from the data as a heatmap. Don't forget to:
68 #
69 #     1)Label the heatmap's rows and columns with the corresponding sport (rotate the x-
    axis labels by 45 degrees so that the labels are readable)
70 #
71 #     2)Add a title to your figure (e.g. similarity)
72 #
73 #     3)Add a colorbar. Limit the colobar values between 0 and 1.
74 #
75 #     4)Use default colormap
76 #
77 #     5)Upload figure PS4_Q1.png to gradescope.
78 #
79 # Hint - look up matplotlib's imshow.
80
81 # In[155]:
82
83
84 #YOUR CODE HERE
85 figure, ax = plt.subplots()
```



```
86 heatmap = ax.imshow(similarity, interpolation='nearest')
87 ax.set_xticks(range(len(sport_names)))
88 ax.set_yticks(range(len(sport_names)))
89 ax.set_xticklabels(sport_names)
90 ax.set_yticklabels(sport_names)
91
92 plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
93         rotation_mode="anchor")
94 plt.title("Psychological similarity between sports")
95
96 plt.colorbar(heatmap)
97 figure.set_size_inches(10, 10)
98
99 figure.savefig('PS4_Q1.png')
100
101
102 # ## Q2. Distance [2 pts, SOLO]
103 #
104 # To implement MDS, we need a measure of psychological **distance**. The dataset includes
105 # measures of similarity, not distance.
106 #
107 # Here we will use  $d = 1 - s$  as a method to transform similarity to distance.
108 #
109 # Write a function that converts all similarity measures in the dataset into distances,
110 # using the above provided transformation method. Function should return the output called
111 # distance (Hint: this variable will be used as an input in some of the functions you'll write
112 # in the following questions).
113 #
114 # Plot a scatterplot of the dataset's distances (x axis) against their similarity (y axis).
115 # Label your figure.
116 #
117 # Upload figure PS4_Q2.png to gradescope.
118 #
```

```
115 # In[159]:
116
117
118 #YOUR CODE HERE
119 4 def convert_similarity_to_distance(similarity):
120     distance = np.zeros(similarity.shape)
121     for i in range(len(similarity)):
122         for j in range(len(similarity[0])):
123             distance[i][j] = 1 - similarity[i][j]
124
125     return distance
126
127 distance = convert_similarity_to_distance(similarity)
128
129 distance_1d = np.reshape(distance, len(distance) * len(distance[0]))
130 similarity_1d = np.reshape(similarity, len(similarity) * len(similarity[0]))
131
132 figure, ax = plt.subplots()
133
134 plt.scatter(distance_1d, similarity_1d)
135 plt.xlabel('Distance')
136 plt.ylabel('Similarity')
137 plt.title('Dataset distances vs. similarities')
138
139 figure.set_size_inches(10, 6)
140
141 figure.savefig('PS4_Q2.png')
142
143
144 # ## Q3. Stress [5 pts, SOLO]
145 #
```

```

146 # To perform MDS, we will try to find, for each sport i, a position  $p_i=(x_i,y_i)$  in the
    2d space that captures the participants' similarities. To do so, we will build an algorithm
    that minimizes the stress. We'll define stress slightly differently than in class- the
    squared difference between psychological distance  $\psi_{ij} = (1-s_{ij})$  and the MDS
    distance in 2D space:
147 #
148 #  $\text{Stress} \ S = \sum_{i>j} (\psi_{ij} - \text{dist}(p_i, p_j))^2$ 
149 #
150 # Where  $\psi$  is the psychological distance between sport i and sport j that was reported
    by subjects, and  $\text{dist}(p_i, p_j)$  corresponds to the **Euclidean distance**:  $\sqrt{(x_i-x_j)^2 + (y_i-y_j)^2}$ 
151 #
152 # Write a function that computes the Euclidean Distance between two points  $p_1$  and  $p_2$ .
    Then, write a function that takes a  $(n,2)$  ( $n$ =number of sports) matrix of  $(x,y)$  positions
    for each sport, and computes the stress based on the equation above, using your Euclidean
    Distance function.
153 #
154 # Copy the StressCalc function into gradescope.
155
156 # In[143]:
157
158
159 def EuclideanDistance(p1,p2):
160     ''' Takes positions defined by p1 and p2, and returns a euclidean distance value (single
    number).
161     Implement EQ equation provided in the question. Hint: if  $p_1=p_2$ , the function should
    return the value of 0'''
162     #YOUR CODE HERE
163     return ((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2) ** 0.5
164
165
166 # In[144]:
167
168

```

```

169 def StressCalc(positions, distance):
170     ''' Takes positions (n,2) and (n,n) matrix of distance measures
171     (You will use the distance matrix from Q2).
172     Uses these distances and the EuclideanDistance function above which computes ED based on
    positions
173     to calculate the Stress between psychological and ED distances, according to the
    provided formula.'''
174
175     #YOUR CODE HERE
176     stress = 0
177
178     for i in range(len(distance)):
179         for j in range(i, len(distance)):
180             stress += ((distance[i][j] - EuclideanDistance(positions[i], positions[j]))) **
181                        2
182
183     return stress
184
185
186 # In[145]:
187
188
189 # Test case!
190 '''
191 Test case for StressCalc: create an array of positions, where each entry is 1.
192 Use this positions matrix and distance matrix from Q2 to call StressCalc function
193 '''
194
195 positions = np.ones((len(similarity),2))
196 print(['Stress value should be 111.57. Output stress value is: ' +
197        str(StressCalc(positions,distance))])
198

```

```
199 # ## Q4. Gradient [10 pts, HELP]
200 # To minimize the stress, we will numerically compute the gradient using a multidimensional
    version of the simple rule for derivatives:
201 #
202 #  $\frac{df}{dp}(p) = \frac{f(p+\delta) - f(p-\delta)}{2\delta}$ 
203 #
204 # where  $\delta$  takes on a small value, and  $f$  is the stress function you wrote in the
    previous question. To compute the gradient, we will compute this approximate derivative with
    respect to each coordinate of each point.
205 #
206 # Write a function that takes an n-by-2 matrix (n=number of sports) of (x,y) positions for
    each sport and computes the gradient (i.e. applies the numerical rule above to each
    coordinate location). This should return an n-by-2 gradient matrix.
207 #
208 #
209 # Use  $\delta = 0.01$ 
210 #
211 # Copy your code into gradescope.
212 #
213
214 # In[180]:
215
216
217 delta = .01
218 positions = np.random.rand(len(similarity),2)
219
220 #YOUR CODE HERE
221 def gradient_at_point(positions, i, j, delta):
222     plus_delta, minus_delta = positions.copy(), positions.copy()
223
224     plus_delta[i][j] += delta
225     minus_delta[i][j] -= delta
226
```



```
227     return (StressCalc(plus_delta, distance) - StressCalc(minus_delta, distance)) / (2 *
    delta)
228
229 def compute_gradient(positions, delta):
230     gradient_matrix = np.zeros(positions.shape)
231     for i in range(len(gradient_matrix)):
232         for j in range(len(gradient_matrix[0])):
233             gradient_matrix[i][j] = gradient_at_point(positions, i, j, delta)
234
235     return gradient_matrix
236
237
238 # ## Q5.1 MDS [10 pts, HELP]
239 #
240 # Write the MDS code: the code that follows a gradient in order to find positions that
    minimize the stress. Start from a random position, and be sure to take small steps in the
    direction of the gradient (e.g.  $\alpha$ *gradient, with step size  $\alpha=0.01$ ), to find a set of
    positions that minimizes the stress. Use 100 steps of gradient descent.
241 #
242 # Copy your code in gradescope.
243
244 # In[181]:
245
246
247 #YOUR CODE HERE
248 STEP_SIZE = 0.01
249 print(f'Initial Positions: \n {positions}')
250 steps, stress_values = [], []
251
252 for iteration in range(100):
253     positions -= (compute_gradient(positions, delta) * STEP_SIZE)
254     step_no, stress_value = iteration + 1, StressCalc(positions, distance)
255     print(f'Iteration #{step_no}, Total Stress: {stress_value}')
256     steps.append(step_no)
```

```
257     stress_values.append(stress_value)
258
259 print(f'Final Positions: \n {positions}')
260
261
262 # ## Q5.2 [5 pts, SOLO]
263 #
264 # Plot the names of sports at the resulting coordinates. Hint: look up axis.text for plotting
    the sports names.
265 #
266 # Upload PS4_Q5_2.png in gradescope.
267
268 # In[182]:
269
270
271 #YOUR CODE HERE
272 pos = 0
273 figure, ax = plt.subplots()
274
275 for sport in positions:
276     plt.plot(sport[0], sport[1], '.')
277     plt.text(sport[0], sport[1] + 0.01, sport_names[pos])
278     pos += 1
279
280 plt.title('Plot of Sports at Locations of Minimized Stress')
281 plt.xlabel('Final x-position')
282 plt.ylabel('Final y-position')
283
284 figure.set_size_inches(10, 6)
285 figure.savefig('PS4_Q5_2.png')
286
287
288 # ## Q5.3 [5 pts, SOLO]
289 # Plot the stress as a function of step number (x axis = step number, y axis= stress).
```

```
290 #
291 # Upload PS4_Q5_3 in gradescope.
292
293 # In[183]:
294
295
296 #YOUR CODE HERE
297 figure, ax = plt.subplots()
298
299 plt.plot(steps, stress_values, '.')
300 plt.xlabel('Step Number')
301 plt.ylabel('Stress Value')
302 plt.title('Stress Plotted as a Function of Step Number')
303
304 figure.set_size_inches(10, 6)
305 figure.savefig('PS4_Q5_3.png')
306
307
308 # ## Q6. Validation [5pts, SOLO]
309 #
310 # Make a scatter plot of the distances obtained by running your MDS function vs. people's
    reported distances *d=(1-s)*.
311 #
312 # Upload PS4_Q6.png to gradescope.
313 #
314 # Briefly describe what a good and bad MDS-psychological distance relationship would look
    like, and whether yours is good or bad. Enter your response in gradescope.
315 #
316
317 # In[184]:
318
319
320 #YOUR CODE HERE
321 figure, ax = plt.subplots()
```



```
322
323 MDS_distances = np.zeros(distance.shape)
324
325 for i in range(len(positions)):
326     for j in range(len(positions)):
327         MDS_distances[i][j] = EuclideanDistance(positions[i], positions[j])
328
329
330 plt.scatter(MDS_distances, distance, s=10, c='green')
331 plt.xlabel('Distances obtained by running MDS')
332 plt.ylabel('Reported distances (d = 1-s)')
333 plt.title('MDS Distances Plotted Against Reported Distances')
334
335 figure.set_size_inches(10, 6)
336 figure.savefig('PS4_Q6.png')
337
338
339 # Intuitively, it makes sense for a good MDS-psychological distance relationship to show the
    points lying along some sort of trend line, because this indicates that distances obtained
    by running our MDS function are closely related to people's reported distances. In other
    words, the points on the scatter plot should have similar x- and y-values and be pointing to
    the top right, indicating a positive correlation. From the scatter plot, we can see that our
    relationship doesn't seem too bad, though there doesn't seem to be any strong positive
    correlation. That being said, the points aren't randomly distributed and seem to follow a
    very weak positive trend, which is a good sign.
340
341 # ## Q7.1 Iterating MDS [3pts, SOLO]
342 #
343 # Run your MDS code 9 times, and plot the corresponding final positions in a figure with
    subplots in a 3x3 grid. Indicate the code iteration number in each subplot title. Scale the
    figure size using figsize=(15,15).
344 #
345 # Are they all the same or not? Why? Enter your response in gradescope.
346 #
```

```
347 # Upload PS4_Q7_1.png in gradescope.
348
349 # In[189]:
350
351
352 #YOUR CODE HERE
353 figure, ax = plt.subplots(3, 3, sharex='row', sharey='row')
354 delta, STEP_SIZE = 0.01, 0.01
355 final_stress_values = []
356 iter_no = 1
357
358 for i in range(3):
359     for j in range(3):
360         positions = np.random.rand(len(similarity), 2)
361         for iteration in range(100):
362             positions -= (compute_gradient(positions, delta) * STEP_SIZE)
363             final_stress_values.append(StressCalc(positions, distance))
364
365         pos = 0
366         for sport in positions:
367             ax[i][j].plot(sport[0], sport[1], '.')
368             ax[i][j].text(sport[0], sport[1] + 0.01, sport_names[pos])
369             pos += 1
370
371         ax[i][j].set_title(f'Plot of Sports at Final Positions, Iteration #{iter_no}')
372         ax[i][j].set_xlabel('Final x-position')
373         ax[i][j].set_ylabel('Final y-position')
374
375         iter_no += 1
376
377 figure.set_size_inches(15, 15)
378 figure.savefig('PS4_Q7_1.png')
379
380
```

```
381 # These plots are not all the same. This is expected however, since we are starting at
    # random initial positions to begin with. This means that when we perform gradient descent, we
    # will almost surely end with different final values that minimize the final stress value.
382
383 # ## Q7.2 Best representation [3pts, SOLO]
384 # In another figure, plot the final stress value as a function of the MDS iteration (9) in
    # the previous question. If you wanted to pick the best final representation based on this
    # plot, how would you do it? What criteria would you use? Which iteration is your best?
385 #
386 # Enter your answer in gradescope.
387 #
388 # Upload PS4_Q7_2.png.
389
390 # In[199]:
391
392
393 #YOUR CODE HERE
394 figure, ax = plt.subplots()
395
396 plt.plot([i+1 for i in range(9)], final_stress_values, 'bo')
397 plt.title('Final Stress Value Plotted as a Function of Iteration #')
398 plt.xlabel('Iteration Number')
399 plt.ylabel('Final Stress Value')
400
401 figure.set_size_inches(10, 6)
402 figure.savefig('PS4_Q7_2.png')
403
404
405 # If I wanted to pick the best final representation, I would pick the iteration which
    # resulted in the minimum final stress value. The stress function is our equivalent of a loss
    # function, which indicates how far our representation is from the optimal stress value for
    # our initial sport positions. A low stress value indicates that our measure of "similarity"
    # between two sports is close to the given similarity values. Iteration 1 is my best.
406
```

```
407 # ## Q7.3 [4pts, SOLO]
408 # Do your best results agree with your intuitions about how this domain might be organized?
    Why or why not? Answer in 2-3 sentences.
409 #
410 # Enter your response in gradescope.
411 #
412
413 # Yes, my best results with my intuitions of how these sports should be organized. We can
    notice some type of clustering happening here, where ball sports like volleyball, softball,
    and and basketball have very similar final positions, and other sports which are not very
    similar (like track and boxing) are located far from one another. This is interesting
    because clustering is the natural way to organize sports like these, i.e. we have categories
    like 'ball sports' and 'water sports' for a reason.
414
415 # ## Q8 [5pts, SOLO]
416 #
417 # Run MDS 2 times, with 2 different step sizes ( $\alpha=.02$  and  $\alpha=.05$ ). Plot Stress over time
    for each run in the same plot. Don't forget to add a legend, labeling which MDS step size the
    line refers to, in addition to the usual axis labels and title. What happens if you use a
    bigger step in your MDS? Why?
418 #
419 #
420 # Enter your answer in gradescope.
421 #
422 # Upload PS4_Q8.png.
423
424 # In[208]:
425
426
427 #YOUR CODE HERE
428 figure, ax = plt.subplots()
429 delta = 0.01
430 init_positions = np.random.rand(len(similarity), 2)
431
```

```
432 # Step size = 0.02
433 positions = init_positions.copy()
434 steps, stress_values = [], []
435 STEP_SIZE = 0.02
436
437 for iteration in range(100):
438     positions -= (compute_gradient(positions, delta) * STEP_SIZE)
439     step_no, stress_value = iteration + 1, StressCalc(positions, distance)
440     steps.append(step_no)
441     stress_values.append(stress_value)
442
443 ax.plot(steps, stress_values, color='blue')
444
445 # Step size = 0.05
446 positions = init_positions.copy()
447 steps, stress_values = [], []
448 STEP_SIZE = 0.05
449
450 for iteration in range(100):
451     positions -= (compute_gradient(positions, delta) * STEP_SIZE)
452     step_no, stress_value = iteration + 1, StressCalc(positions, distance)
453     steps.append(step_no)
454     stress_values.append(stress_value)
455
456 ax.plot(steps, stress_values, color='red')
457
458 plt.title('Stress Plotted Over Time')
459 plt.xlabel('Step Number')
460 plt.ylabel('Stress Value')
461 plt.legend(['$a = 0.02$', '$a = 0.05$'])
462
463 figure.set_size_inches(10, 6)
464 figure.savefig('PS4_Q8.png')
465
```

```
466
467 # If we use a bigger step size, we see that the plot of stress over time follows a much less
    well defined path, and seems to actually work against finding the optimal stress value. This
    is because a larger step size may not allow gradient descent to converge properly and find a
    global minimum of the overall stress function, causing our values of stress over time to
    vastly oscillate and differ as we perform more iterations. This is why determining the
    'ideal' step size when performing gradient descent is so integral, as it may be the
    difference between convergence and non-convergence.
468
469 # <div style="background-color: #c1f2a5">
470 #
471 # # Submission
472 #
473 # When you're done with your problem set, do the following:
474 # - Upload your answers in Gradescope's PS4.
475 # - Upload your code as .py file in PS4-code in Gradescope (To convert the notebook into .py
    file click on File > Download as > Python (.py)).
476 #
477 #
478 #
479 #
480 # </div>
481
482 # In[ ]:
483
484
485
486
487
```

Jon Gill PS4_Jon_Gill.py

```
1 #!/usr/bin/env python
2 # coding: utf-8
```

```
3
4 # <div style="background-color: #c1f2a5">
5 #
6 #
7 # # PS4
8 #
9 # In this problem set, you will implement multidimensional scaling (MDS) from scratch. You
  may use standard matrix/vector libraries (e.g. numpy) but you must implement two dimensional
  MDS itself on your own and not use an existing software package. MDS attempts to find an
  arrangement of points such that the distances between points match human-judged
  similarities.
10 #
11 # # Instructions
12 #
13 #
14 #
15 # Remember to do your problem set in Python 3. Fill in `#YOUR CODE HERE`.
16 #
17 # Make sure:
18 # - that all plots are scaled in such a way that you can see what is going on (while still
  respecting specific plotting instructions)
19 # - that the general patterns are fairly represented.
20 # - to label all x- and y-axes, and to include a title.
21 #
22 # </div>
23
24 # In[1]:
25
26
27 import numpy as np
28 import matplotlib.pyplot as plt
29 # to import the data set
30 from scipy.io import loadmat
31
```

```
32
33 # ## Import and examine data
34 #
35 # We will be using a data set from Romney, A. K., Brewer, D. D., & Batchelder, W. H. (1993).
    Predicting Clustering from Semantic Structure. Psychological Science, 4(1), 28-34, via
    https://faculty.sites.uci.edu/mdlee/similarity-data/. The data set is saved in
    PS4_dataset.mat, and includes pairwise similarity measures between 21 sports. Make sure that
    the PS notebook and the data set are in the same directory!
36 #
37 # As our first step, we will download and examine the data:
38 #
39
40 # In[2]:
41
42
43 data_set = loadmat('PS4_dataset.mat')
44 similarity = data_set['similarity']
45 sport_names = data_set['sport_names']
46
47
48 # As we can see, our data contains information for 21 different sports as listed below:
49
50 # In[3]:
51
52
53 print(sport_names)
54
55
56 # We also have a similarity matrix for each sport, which gives us the psychological
    similarity of that sport with all the other sports in the data:
57
58 # In[4]:
59
60
```



```
61 #Look at the first similarity matrix, which corresponds to football's similarity with itself
    and all other sports
62 print(similarity[0])
63
64
65 # ## Q1. Visualize similarity [5pts, HELP]
66
67 # Plot the "similarity" measures from the data as a heatmap. Don't forget to:
68 #
69 #     1)Label the heatmap's rows and columns with the corresponding sport (rotate the x-
    axis labels by 45 degrees so that the labels are readable)
70 #
71 #     2)Add a title to your figure (e.g. similarity)
72 #
73 #     3)Add a colorbar. Limit the colorbar values between 0 and 1.
74 #
75 #     4)Use default colormap
76 #
77 #     5)Upload figure PS4_Q1.png to gradescope.
78 #
79 # Hint - look up matplotlib's imshow.
80
81 # In[32]:
82
83
84 figure, ax = plt.subplots()
85 heatmap = ax.imshow(similarity, interpolation='nearest')
86 num_sports = len(sport_names)
87 ax.set_xticks(range(num_sports))
88 ax.set_xticklabels(sport_names)
89 ax.set_yticks(range(num_sports))
90 ax.set_yticklabels(sport_names)
91 ax.set_title("Psychological Similarity Between Sports")
92 labels = ax.get_xticklabels()
```

```
93 heat_map = ax.imshow(similarity, interpolation='nearest')
94
95 plt.setp(labels, rotation=45, ha="right", rotation_mode="anchor")
96 plt.colorbar(heat_map)
97
98 figure.set_size_inches(15, 15)
99 figure.savefig('PS4_Q1.png')
100
101
102 # ## Q2. Distance [2 pts, SOLO]
103 #
104 # To implement MDS, we need a measure of psychological **distance**. The dataset includes
    measures of similarity, not distance.
105 #
106 # Here we will use  $d = 1 - s$  as a method to transform similarity to distance.
107 #
108 # Write a function that converts all similarity measures in the dataset into distances,
    using the above provided transformation method. Function should return the output called
    distance (Hint: this variable will be used as an input in some of the functions you'll write
    in the following questions).
109 #
110 # Plot a scatterplot of the dataset's distances (x axis) against their similarity (y axis).
    Label your figure.
111 #
112 # Upload figure PS4_Q2.png to gradescope.
113 #
114
115 # In[30]:
116
117
118 def similarity_to_distance(similarity):
119     distance = np.zeros(similarity.shape)
120     for i in range(len(similarity)):
121         for j in range(len(similarity[0])):
```

4

```
122         distance[i][j] = 1 - similarity[i][j]
123     return distance
124
125 distance = similarity_to_distance(similarity)
126 distance_flattened = distance.flatten()
127 similarity_flattened = similarity.flatten()
128
129 figure, ax = plt.subplots()
130 plt.scatter(distance_flattened, similarity_flattened)
131 ax.set_xlabel('Distance')
132 ax.set_ylabel('Similarity')
133 ax.set_title('Distances vs. Similarities')
134
135 figure.set_size_inches(10, 10)
136 figure.savefig('PS4_Q2.png')
137
138
139 # ## Q3. Stress [5 pts, SOLO]
140 #
141 # To perform MDS, we will try to find, for each sport i, a position  $p_i=(x_i,y_i)$  in the
142 # 2d space that captures the participants' similarities. To do so, we will build an algorithm
143 # that minimizes the stress. We'll define stress slightly differently than in class- the
144 # squared difference between psychological distance  $\psi_{ij} = (1-s_{ij})$  and the MDS
145 # distance in 2D space:
146 #
147 # 
$$\mathrm{Stress} \ S = \sum_{i \neq j} (\psi_{ij} - \mathrm{dist}(p_i, p_j))^2$$

148 #
149 # Where  $\psi$  is the psychological distance between sport i and sport j that was reported
150 # by subjects, and  $\mathrm{dist}(p_i, p_j)$  corresponds to the **Euclidean distance**:  $\sqrt{(x_i-x_j)^2 + (y_i-y_j)^2}$ 
151 #
```

```
147 # Write a function that computes the Euclidean Distance between two points $p_1$ and $p_2$.  
    Then, write a function that takes a $(n,2)$ (n=number of sports) matrix of $(x,y)$ positions  
    for each sport, and computes the stress based on the equation above, using your Euclidean  
    Distance function.  
148 #  
149 # Copy the StressCalc function into gradescope.  
150  
151 # In[33]:  
152  
153  
154 def EuclideanDistance(p1,p2):  
155     ''' Takes positions defined by p1 and p2, and returns a euclidean distance value (single  
        number).  
156     Implement EQ equation provided in the question. Hint: if p1=p2, the function should  
        return the value of 0'''  
157     return pow(pow((p1[0] - p2[0]), 2) + pow((p1[1] - p2[1]), 2), 0.5)  
158  
159  
160 # In[37]:  
161  
162  
163 def StressCalc(positions, distance):  
164     ''' Takes positions (n,2) and (n,n) matrix of distance measures  
165     (You will use the distance matrix from Q2).  
166     Uses these distances and the EuclideanDistance function above which computes ED based on  
        positions  
167     to calculate the Stress between psychological and ED distances, according to the  
        provided formula.'''  
168     stress = 0  
169     for i in range(len(distance)):  
170         for j in range(i, len(distance)):  
171             d = EuclideanDistance(positions[i], positions[j])  
172             stress += pow((distance[i][j] - d), 2)  
173     return stress
```

```
174
175
176 # In[38]:
177
178
179 # Test case!
180 '''
181 Test case for StressCalc: create an array of positions, where each entry is 1.
182 Use this positions matrix and distance matrix from Q2 to call StressCalc function
183 '''
184
185 positions = np.ones((len(similarity),2))
186 print(['Stress value should be 111.57. Output stress value is: ' +
187        str(StressCalc(positions,distance))])
188
189 # ## Q4. Gradient [10 pts, HELP]
190 # To minimize the stress, we will numerically compute the gradient using a multidimensional
191 # version of the simple rule for derivatives:
192 #
193 #  $\frac{df}{dp}(p) = \frac{f(p+\delta) - f(p-\delta)}{2\delta}$ 
194 # where  $\delta$  takes on a small value, and  $f$  is the stress function you wrote in the
195 # previous question. To compute the gradient, we will compute this approximate derivative with
196 # respect to each coordinate of each point.
197 #
198 # Write a function that takes an n-by-2 matrix (n=number of sports) of (x,y) positions for
199 # each sport and computes the gradient (i.e. applies the numerical rule above to each
200 # coordinate location). This should return an n-by-2 gradient matrix.
201 #
202 # Use  $\delta = 0.01$ 
203 #
204 # Copy your code into gradescope.
```

```
202 #
203
204 # In[50]:
205
206
207 delta = .01
208 positions = np.random.rand(len(similarity),2)
209
210 def compute_gradient(positions, delta):
211     gradient_matrix = np.zeros(positions.shape)
212     for i in range(len(gradient_matrix)):
213         for j in range(len(gradient_matrix[0])):
214             plus_delta = positions.copy()
215             minus_delta = positions.copy()
216             plus_delta[i][j] += delta
217             minus_delta[i][j] -= delta
218             diff = (StressCalc(plus_delta, distance) - StressCalc(minus_delta, distance))
219             gradient_matrix[i][j] = diff / (2 * delta)
220     return gradient_matrix
221
222
223 # ## Q5.1 MDS [10 pts, HELP]
224 #
225 # Write the MDS code: the code that follows a gradient in order to find positions that
226 # minimize the stress. Start from a random position, and be sure to take small steps in the
227 # direction of the gradient (e.g.  $\alpha$ *gradient, with step size  $\alpha=0.01$ ), to find a set of
228 # positions that minimizes the stress. Use 100 steps of gradient descent.
229 #
230 # Copy your code in gradescope.
231
232 # In[51]:
233
234 alpha = 0.01
```

```
233 stress_vals = []
234 for iteration in range(1, 101):
235     positions -= (compute_gradient(positions, delta) * alpha)
236     stress_val = StressCalc(positions, distance)
237     stress_vals.append(stress_val)
238
239
240 # ## Q5.2 [5 pts, SOLO]
241 #
242 # Plot the names of sports at the resulting coordinates. Hint: look up axis.text for plotting
    the sports names.
243 #
244 # Upload PS4_Q5_2.png in gradescope.
245
246 # In[56]:
247
248
249 idx = 0
250 figure, ax = plt.subplots()
251
252 for sport in positions:
253     ax.plot(sport[0], sport[1], '.')
254     ax.text(sport[0], sport[1] + 0.01, sport_names[idx])
255     idx += 1
256
257 ax.set_title('Sports at Locations of Minimized Stress')
258 ax.set_xlabel('Final x-position')
259 ax.set_ylabel('Final y-position')
260
261 figure.set_size_inches(10, 10)
262 figure.savefig('PS4_Q5_2.png')
263
264
265 # ## Q5.3 [5 pts, SOLO]
```

```
266 # Plot the stress as a function of step number (x axis = step number, y axis= stress).
267 #
268 # Upload PS4_Q5_3 in gradescope.
269
270 # In[60]:
271
272
273 figure, ax = plt.subplots()
274
275 ax.plot(stress_values, '.')
276 ax.set_xlabel('Step Number')
277 ax.set_ylabel('Stress')
278 ax.set_title('Stress vs. Step Number')
279
280 figure.set_size_inches(10, 10)
281 figure.savefig('PS4_Q5_3.png')
282
283
284 # ## Q6. Validation [5pts, SOLO]
285 #
286 # Make a scatter plot of the distances obtained by running your MDS function vs. people's
    reported distances *d=(1-s)*.
287 #
288 # Upload PS4_Q6.png to gradescope.
289 #
290 # Briefly describe what a good and bad MDS-psychological distance relationship would look
    like, and whether yours is good or bad. Enter your response in gradescope.
291 #
292
293 # In[67]:
294
295
296 figure, ax = plt.subplots()
297
```



```
298 mds_distances = np.zeros(distance.shape)
299
300 for i in range(len(positions)):
301     for j in range(len(positions)):
302         mds_distances[i][j] = EuclideanDistance(positions[i], positions[j])
303
304 ax.scatter(mds_distances, distance, s=10, c='blue')
305 ax.set_xlabel('Distances from running MDS')
306 ax.set_ylabel('Reported distances  $(d = 1-s)$ ')
307 ax.set_title('MDS Distances vs. Reported Distances')
308
309 figure.set_size_inches(10, 10)
310 figure.savefig('PS4_Q6.png')
311
312
313 # ## Q7.1 Iterating MDS [3pts, SOLO]
314 #
315 # Run your MDS code 9 times, and plot the corresponding final positions in a figure with
316 # subplots in a 3x3 grid. Indicate the code iteration number in each subplot title. Scale the
317 # figure size using figsize=(15,15).
318 #
319 # Are they all the same or not? Why? Enter your response in gradescope.
320 #
321 # Upload PS4_Q7_1.png in gradescope.
322 #
323 # In[81]:
324
325 figure, ax = plt.subplots(3, 3, sharex='row', sharey='row')
326 delta = 0.01
327 alpha = 0.01
328 final_stress_vals = []
329 iteration = 1
```

```
330 for i in range(3):
331     for j in range(3):
332         positions = np.random.rand(len(similarity), 2)
333         for iteration in range(100):
334             positions -= (compute_gradient(positions, delta) * alpha)
335         final_stress_vals.append(StressCalc(positions, distance))
336
337     idx = 0
338     for sport in positions:
339         ax[i][j].plot(sport[0], sport[1], '.')
340         ax[i][j].text(sport[0], sport[1] + 0.01, sport_names[idx])
341         idx += 1
342
343     ax[i][j].set_title(f'Plot of Sports at Final Positions, Iteration #{iteration}')
344     ax[i][j].set_xlabel('Final x-position')
345     ax[i][j].set_ylabel('Final y-position')
346
347     iter_no += 1
348
349 figure.set_size_inches(15, 15)
350 figure.savefig('PS4_Q7_1.png')
351
352
353 # ## Q7.2 Best representation [3pts, SOLO]
354 # In another figure, plot the final stress value as a function of the MDS iteration (9) in
355 # the previous question. If you wanted to pick the best final representation based on this
356 # plot, how would you do it? What criteria would you use? Which iteration is your best?
357 #
358 # Enter your answer in gradescope.
359 #
360 # Upload PS4_Q7_2.png.
361
362 # In[76]:
```

```
362
363 figure, ax = plt.subplots()
364
365 ax.plot([i for i in range(1,10)], final_stress_values, 'go')
366 ax.set_title('Final Stress Value vs. Iteration #')
367 ax.set_xlabel('Iteration #')
368 ax.set_ylabel('Final Stress Value')
369
370 figure.set_size_inches(10, 10)
371 figure.savefig('PS4_Q7_2.png')
372
373
374 # ## Q7.3 [4pts, SOLO]
375 # Do your best results agree with your intuitions about how this domain might be organized?
    Why or why not? Answer in 2-3 sentences.
376 #
377 # Enter your response in gradescope.
378 #
379
380 # ## Q8 [5pts, SOLO]
381 #
382 # Run MDS 2 times, with 2 different step sizes ( $\alpha=.02$  and  $\alpha=.05$ ). Plot Stress over time
    for each run in the same plot. Don't forget to add a legend, labeling which MDS step size the
    line refers to, in addition to the usual axis labels and title. What happens if you use a
    bigger step in your MDS? Why?
383 #
384 #
385 # Enter your answer in gradescope.
386 #
387 # Upload PS4_Q8.png.
388
389 # In[80]:
390
391
```

```
392 figure, ax = plt.subplots()
393 delta = 0.01
394 init_positions = np.random.rand(len(similarity), 2)
395
396 positions = init_positions.copy()
397 stress_vals = []
398 alpha = 0.02
399
400 for iteration in range(1, 101):
401     positions -= (compute_gradient(positions, delta) * alpha)
402     stress_val = StressCalc(positions, distance)
403     stress_vals.append(stress_val)
404
405 ax.plot(stress_vals, color='green')
406
407 positions = init_positions.copy()
408 stress_vals = []
409 alpha = 0.05
410
411 for iteration in range(1, 101):
412     positions -= (compute_gradient(positions, delta) * alpha)
413     stress_val = StressCalc(positions, distance)
414     stress_vals.append(stress_val)
415
416 ax.plot(stress_vals, color='red')
417
418 ax.set_title('Stress Plotted Over Time')
419 ax.set_xlabel('Step Number')
420 ax.set_ylabel('Stress Value')
421 plt.legend(['$a = 0.02$', '$a = 0.05$'])
422
423 figure.set_size_inches(10, 10)
424 figure.savefig('PS4_Q8.png')
425
```

```
426
427 # <div style="background-color: #c1f2a5">
428 #
429 # # Submission
430 #
431 # When you're done with your problem set, do the following:
432 # - Upload your answers in Gradescope's PS4.
433 # - Upload your code as .py file in PS4-code in Gradescope (To convert the notebook into .py
    file click on File > Download as > Python (.py)).
434 #
435 #
436 #
437 #
438 # </div>
439
440 # In[ ]:
441
442
443
444
445
```