CogSci131 Assignment2

July, 2019

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import distance
names = ["football","baseball","basketball","tennis","softball","canoeing","handball"
```

In [13]:
```
# load the csv provided on bcourses
similarities = np.loadtxt(open("similarities.csv", "rb"), delimiter=",", skiprows=1)
similarities
#array.shape will give the dimension for np array which is 21*21
```

Out[13]:
```
array([[1.        , 0.18518519, 0.48148148, 0.14814815, 0.74074074,
        0.07407407, 0.77777778, 0.77777778, 0.88888889, 0.11111111,
        0.14814815, 0.40740741, 0.85185185, 0.92592593, 0.7037037 ,
        0.03703704, 0.        , 0.74074074, 0.07407407, 0.48148148,
        0.03703704],
       [0.18518519, 1.        , 0.33333333, 0.18518519, 0.96296296,
        0.        , 1.        , 0.92592593, 0.59259259, 0.62962963,
        0.        , 0.40740741, 0.22222222, 0.37037037, 0.81481481,
        0.        , 0.7037037 , 0.77777778, 0.11111111, 0.07407407,
        0.        ],
       [0.48148148, 0.33333333, 1.        , 0.59259259, 0.88888889,
        0.14814815, 0.77777778, 0.74074074, 0.62962963, 0.18518519,
        0.11111111, 0.07407407, 0.66666667, 0.88888889, 0.40740741,
        0.03703704, 0.14814815, 0.14814815, 0.03703704, 0.22222222,
        0.18518519],
       [0.14814815, 0.18518519, 0.59259259, 1.        , 0.88888889,
        0.11111111, 0.81481481, 0.51851852, 0.44444444, 0.14814815,
        0.44444444, 0.40740741, 0.14814815, 0.44444444, 0.14814815,
        0.18518519, 0.85185185, 0.62962963, 0.07407407, 0.18518519,
        0.2962963 ],
       [0.74074074, 0.96296296, 0.88888889, 0.88888889, 1.        ,
        0.        , 0.51851852, 0.07407407, 0.81481481, 0.48148148,
        0.03703704, 0.03703704, 0.11111111, 0.74074074, 0.33333333,
        0.03703704, 0.25925926, 0.66666667, 0.07407407, 0.22222222,
        0.11111111],
       [0.07407407, 0.        , 0.14814815, 0.11111111, 0.        ,
        1.        , 0.11111111, 0.07407407, 0.03703704, 0.07407407,
        0.81481481, 0.37037037, 0.18518519, 0.03703704, 0.11111111,
```

```
 0.62962963, 0.33333333, 0.22222222, 0.88888889, 0.25925926,
 0.11111111],
[0.77777778, 1.        , 0.77777778, 0.81481481, 0.51851852,
 0.11111111, 1.        , 0.11111111, 0.25925926, 0.03703704,
 0.11111111, 0.44444444, 0.11111111, 0.85185185, 0.14814815,
 0.        , 0.07407407, 0.66666667, 0.        , 0.11111111,
 0.18518519],
[0.77777778, 0.92592593, 0.74074074, 0.51851852, 0.07407407,
 0.07407407, 0.11111111, 1.        , 0.62962963, 0.11111111,
 0.03703704, 0.22222222, 0.25925926, 0.18518519, 0.92592593,
 0.14814815, 0.51851852, 0.74074074, 0.        , 0.81481481,
 0.07407407],
[0.88888889, 0.59259259, 0.62962963, 0.44444444, 0.81481481,
 0.03703704, 0.25925926, 0.62962963, 1.        , 0.96296296,
 0.        , 0.        , 0.40740741, 0.48148148, 0.33333333,
 0.33333333, 0.03703704, 0.22222222, 0.03703704, 0.44444444,
 0.07407407],
[0.11111111, 0.62962963, 0.18518519, 0.14814815, 0.48148148,
 0.07407407, 0.03703704, 0.11111111, 0.96296296, 1.        ,
 0.33333333, 0.18518519, 0.40740741, 0.62962963, 0.51851852,
 0.44444444, 0.        , 0.66666667, 0.03703704, 0.2962963 ,
 0.03703704],
[0.14814815, 0.        , 0.11111111, 0.44444444, 0.03703704,
 0.81481481, 0.11111111, 0.03703704, 0.        , 0.33333333,
 1.        , 0.48148148, 0.03703704, 0.14814815, 0.14814815,
 0.92592593, 0.03703704, 0.11111111, 0.96296296, 0.33333333,
 0.77777778],
[0.40740741, 0.40740741, 0.07407407, 0.40740741, 0.03703704,
 0.37037037, 0.44444444, 0.22222222, 0.        , 0.18518519,
 0.48148148, 1.        , 0.07407407, 0.2962963 , 0.51851852,
 0.40740741, 0.25925926, 0.14814815, 0.55555556, 0.11111111,
 0.62962963],
[0.85185185, 0.22222222, 0.66666667, 0.14814815, 0.11111111,
 0.18518519, 0.11111111, 0.25925926, 0.40740741, 0.40740741,
 0.03703704, 0.07407407, 1.        , 0.03703704, 0.11111111,
 0.03703704, 0.07407407, 0.        , 0.07407407, 0.77777778,
 0.2962963 ],
[0.92592593, 0.37037037, 0.88888889, 0.44444444, 0.74074074,
 0.03703704, 0.85185185, 0.18518519, 0.48148148, 0.62962963,
 0.14814815, 0.2962963 , 0.03703704, 1.        , 0.85185185,
 0.03703704, 0.07407407, 0.55555556, 0.07407407, 0.07407407,
 0.33333333],
[0.7037037 , 0.81481481, 0.40740741, 0.14814815, 0.33333333,
 0.11111111, 0.14814815, 0.92592593, 0.33333333, 0.51851852,
 0.14814815, 0.51851852, 0.11111111, 0.85185185, 1.        ,
 0.03703704, 0.44444444, 0.44444444, 0.07407407, 0.51851852,
 0.18518519],
[0.03703704, 0.        , 0.03703704, 0.18518519, 0.03703704,
```

```
 0.62962963, 0.        , 0.14814815, 0.33333333, 0.44444444,
 0.92592593, 0.40740741, 0.03703704, 0.03703704, 0.03703704,
 1.        , 0.11111111, 0.07407407, 0.88888889, 0.03703704,
 0.25925926],
[0.        , 0.7037037 , 0.14814815, 0.85185185, 0.25925926,
 0.33333333, 0.07407407, 0.51851852, 0.03703704, 0.        ,
 0.03703704, 0.25925926, 0.07407407, 0.07407407, 0.44444444,
 0.11111111, 1.        , 0.81481481, 0.07407407, 0.40740741,
 0.18518519],
[0.74074074, 0.77777778, 0.14814815, 0.62962963, 0.66666667,
 0.22222222, 0.66666667, 0.74074074, 0.22222222, 0.66666667,
 0.11111111, 0.14814815, 0.        , 0.55555556, 0.44444444,
 0.07407407, 0.81481481, 1.        , 0.11111111, 0.11111111,
 0.07407407],
[0.07407407, 0.11111111, 0.03703704, 0.07407407, 0.07407407,
 0.88888889, 0.        , 0.        , 0.03703704, 0.03703704,
 0.96296296, 0.55555556, 0.07407407, 0.07407407, 0.07407407,
 0.88888889, 0.07407407, 0.11111111, 1.        , 0.03703704,
 0.33333333],
[0.48148148, 0.07407407, 0.22222222, 0.18518519, 0.22222222,
 0.25925926, 0.11111111, 0.81481481, 0.44444444, 0.2962963 ,
 0.33333333, 0.11111111, 0.77777778, 0.07407407, 0.51851852,
 0.03703704, 0.40740741, 0.11111111, 0.03703704, 1.        ,
 0.62962963],
[0.03703704, 0.        , 0.18518519, 0.2962963 , 0.11111111,
 0.11111111, 0.18518519, 0.07407407, 0.07407407, 0.03703704,
 0.77777778, 0.62962963, 0.2962963 , 0.33333333, 0.18518519,
 0.25925926, 0.18518519, 0.07407407, 0.33333333, 0.62962963,
 1.        ]])
```

In [14]:
```
#1.The dataset provides similarities, not distances. Write down three ways you could c
#similarity to a distance dij and choose one to use in the code. Explain why you chos
#we could use Euclidean, Manhanttan or 1-similarities as distances.  I choose 1-simil
distances=[]
distances=1-similarities
distances
```

Out[14]:
```
array([[0.        , 0.81481481, 0.51851852, 0.85185185, 0.25925926,
        0.92592593, 0.22222222, 0.22222222, 0.11111111, 0.88888889,
        0.85185185, 0.59259259, 0.14814815, 0.07407407, 0.2962963 ,
        0.96296296, 1.        , 0.25925926, 0.92592593, 0.51851852,
        0.96296296],
       [0.81481481, 0.        , 0.66666667, 0.81481481, 0.03703704,
        1.        , 0.        , 0.07407407, 0.40740741, 0.37037037,
        1.        , 0.59259259, 0.77777778, 0.62962963, 0.18518519,
        1.        , 0.2962963 , 0.22222222, 0.88888889, 0.92592593,
        1.        ],
       [0.51851852, 0.66666667, 0.        , 0.40740741, 0.11111111,
```

```
 0.85185185, 0.22222222, 0.25925926, 0.37037037, 0.81481481,
 0.88888889, 0.92592593, 0.33333333, 0.11111111, 0.59259259,
 0.96296296, 0.85185185, 0.85185185, 0.96296296, 0.77777778,
 0.81481481],
[0.85185185, 0.81481481, 0.40740741, 0.        , 0.11111111,
 0.88888889, 0.18518519, 0.48148148, 0.55555556, 0.85185185,
 0.55555556, 0.59259259, 0.85185185, 0.55555556, 0.85185185,
 0.81481481, 0.14814815, 0.37037037, 0.92592593, 0.81481481,
 0.7037037 ],
[0.25925926, 0.03703704, 0.11111111, 0.11111111, 0.        ,
 1.        , 0.48148148, 0.92592593, 0.18518519, 0.51851852,
 0.96296296, 0.96296296, 0.88888889, 0.25925926, 0.66666667,
 0.96296296, 0.74074074, 0.33333333, 0.92592593, 0.77777778,
 0.88888889],
[0.92592593, 1.        , 0.85185185, 0.88888889, 1.        ,
 0.        , 0.88888889, 0.92592593, 0.96296296, 0.92592593,
 0.18518519, 0.62962963, 0.81481481, 0.96296296, 0.88888889,
 0.37037037, 0.66666667, 0.77777778, 0.11111111, 0.74074074,
 0.88888889],
[0.22222222, 0.        , 0.22222222, 0.18518519, 0.48148148,
 0.88888889, 0.        , 0.88888889, 0.74074074, 0.96296296,
 0.88888889, 0.55555556, 0.88888889, 0.14814815, 0.85185185,
 1.        , 0.92592593, 0.33333333, 1.        , 0.88888889,
 0.81481481],
[0.22222222, 0.07407407, 0.25925926, 0.48148148, 0.92592593,
 0.92592593, 0.88888889, 0.        , 0.37037037, 0.88888889,
 0.96296296, 0.77777778, 0.74074074, 0.81481481, 0.07407407,
 0.85185185, 0.48148148, 0.25925926, 1.        , 0.18518519,
 0.92592593],
[0.11111111, 0.40740741, 0.37037037, 0.55555556, 0.18518519,
 0.96296296, 0.74074074, 0.37037037, 0.        , 0.03703704,
 1.        , 1.        , 0.59259259, 0.51851852, 0.66666667,
 0.66666667, 0.96296296, 0.77777778, 0.96296296, 0.55555556,
 0.92592593],
[0.88888889, 0.37037037, 0.81481481, 0.85185185, 0.51851852,
 0.92592593, 0.96296296, 0.88888889, 0.03703704, 0.        ,
 0.66666667, 0.81481481, 0.59259259, 0.37037037, 0.48148148,
 0.55555556, 1.        , 0.33333333, 0.96296296, 0.7037037 ,
 0.96296296],
[0.85185185, 1.        , 0.88888889, 0.55555556, 0.96296296,
 0.18518519, 0.88888889, 0.96296296, 1.        , 0.66666667,
 0.        , 0.51851852, 0.96296296, 0.85185185, 0.85185185,
 0.07407407, 0.96296296, 0.88888889, 0.03703704, 0.66666667,
 0.22222222],
[0.59259259, 0.59259259, 0.92592593, 0.59259259, 0.96296296,
 0.62962963, 0.55555556, 0.77777778, 1.        , 0.81481481,
 0.51851852, 0.        , 0.92592593, 0.7037037 , 0.48148148,
 0.59259259, 0.74074074, 0.85185185, 0.44444444, 0.88888889,
```

```
 0.37037037],
[0.14814815, 0.77777778, 0.33333333, 0.85185185, 0.88888889,
 0.81481481, 0.88888889, 0.74074074, 0.59259259, 0.59259259,
 0.96296296, 0.92592593, 0.        , 0.96296296, 0.88888889,
 0.96296296, 0.92592593, 1.        , 0.92592593, 0.22222222,
 0.7037037 ],
[0.07407407, 0.62962963, 0.11111111, 0.55555556, 0.25925926,
 0.96296296, 0.14814815, 0.81481481, 0.51851852, 0.37037037,
 0.85185185, 0.7037037 , 0.96296296, 0.        , 0.14814815,
 0.96296296, 0.92592593, 0.44444444, 0.92592593, 0.92592593,
 0.66666667],
[0.2962963 , 0.18518519, 0.59259259, 0.85185185, 0.66666667,
 0.88888889, 0.85185185, 0.07407407, 0.66666667, 0.48148148,
 0.85185185, 0.48148148, 0.88888889, 0.14814815, 0.        ,
 0.96296296, 0.55555556, 0.55555556, 0.92592593, 0.48148148,
 0.81481481],
[0.96296296, 1.        , 0.96296296, 0.81481481, 0.96296296,
 0.37037037, 1.        , 0.85185185, 0.66666667, 0.55555556,
 0.07407407, 0.59259259, 0.96296296, 0.96296296, 0.96296296,
 0.        , 0.88888889, 0.92592593, 0.11111111, 0.96296296,
 0.74074074],
[1.        , 0.2962963 , 0.85185185, 0.14814815, 0.74074074,
 0.66666667, 0.92592593, 0.48148148, 0.96296296, 1.        ,
 0.96296296, 0.74074074, 0.92592593, 0.92592593, 0.55555556,
 0.88888889, 0.        , 0.18518519, 0.92592593, 0.59259259,
 0.81481481],
[0.25925926, 0.22222222, 0.85185185, 0.37037037, 0.33333333,
 0.77777778, 0.33333333, 0.25925926, 0.77777778, 0.33333333,
 0.88888889, 0.85185185, 1.        , 0.44444444, 0.55555556,
 0.92592593, 0.18518519, 0.        , 0.88888889, 0.88888889,
 0.92592593],
[0.92592593, 0.88888889, 0.96296296, 0.92592593, 0.92592593,
 0.11111111, 1.        , 1.        , 0.96296296, 0.96296296,
 0.03703704, 0.44444444, 0.92592593, 0.92592593, 0.92592593,
 0.11111111, 0.92592593, 0.88888889, 0.        , 0.96296296,
 0.66666667],
[0.51851852, 0.92592593, 0.77777778, 0.81481481, 0.77777778,
 0.74074074, 0.88888889, 0.18518519, 0.55555556, 0.7037037 ,
 0.66666667, 0.88888889, 0.22222222, 0.92592593, 0.48148148,
 0.96296296, 0.59259259, 0.88888889, 0.96296296, 0.        ,
 0.37037037],
[0.96296296, 1.        , 0.81481481, 0.7037037 , 0.88888889,
 0.88888889, 0.81481481, 0.92592593, 0.92592593, 0.96296296,
 0.22222222, 0.37037037, 0.7037037 , 0.66666667, 0.81481481,
 0.74074074, 0.81481481, 0.92592593, 0.66666667, 0.37037037,
 0.        ]])
```

In [15]:
```
#2.Write the code that follows a gradient in order to find positions that minimize th
```

```
D = 2
N = distances.shape[0]
assert(distances.shape[1] == N and N==len(names))
# Pick a position for each point. Note this is an NxD matrix
# so that pos[11,1] is the y coordinate for the 11th item, coordinate start from 0
# and pos[11] is a (row) vector for the position of the 11th item
pos = np.random.normal(0.0,1.0,size=(N,D))
pos
```

```
Out[15]: array([[ 0.38808236, -0.5117374 ],
               [ 0.4079298 , -0.74066719],
               [-0.35472144,  0.19781971],
               [-0.28252064, -0.36168756],
               [-0.34245253,  0.75021195],
               [ 0.21055759,  0.07102482],
               [-0.19958935,  0.15247627],
               [ 0.72564412, -0.47970665],
               [ 2.42713002, -0.23332432],
               [ 1.95553183,  1.03073615],
               [ 0.20406865,  0.24580944],
               [-1.33661673,  0.36783334],
               [ 0.61386319,  1.10541977],
               [ 0.84254918, -1.35074653],
               [ 2.24115926, -0.47220576],
               [ 1.48042411, -0.69015011],
               [-0.54337685, -1.07991108],
               [ 0.65788642,  0.53321827],
               [-1.40730659,  0.70196234],
               [ 0.55242083,  2.29517766],
               [ 0.14030958,  0.2154872 ]])
```

```
In [16]: def dist(a,b):
             # Compute the Euclidean distance between two locations (numpy arrays) a and b
             # Thus, dist(pos[1], pos[2]) gives the distance between the locations for items 1
             mdps = distance.euclidean(a, b)
             return mdps
```

```
In [17]: def stress(p):
         # Take a matrix of positions (called here "p") and return the stress
             psy_dis=[]
             mdps=[]
             for i in range(0,20):
                 for j in range(i+1, 21):# in total 21 items
                     mdps.append(dist(p[i],p[j]))
                     psy_dis.append(distances[i,j])
             tmp=np.array(psy_dis)-np.array(mdps)
             stress=np.sum(tmp**2)
             print(stress)
```

```
        return stress
```

```
In [18]: def add_delta(p, i, d, delta):
             # This is a helper function that will make a new vector which is the same as p (a
             # p[i,d] has been increased by delta (which may be positive or negative)
             v = np.array(p)
             v[i, d] += delta
             return v


         def subtract_delta(p, i, d, delta):
             # This is a helper function that will make a new vector which is the same as p (a
             # p[i,d] has been increased by delta (which may be positive or negative)
             v = np.array(p)
             v[i, d] -= delta
             return v

         def compute_gradient(p, i,d, delta = 0.001):
             # compute the gradient of the stress function with repect to the [i,d] entry of a
             # (e.g. the derivative of stress with respect to the i'th coordinate of the x'th
             # Here, to compute numerically, you can use the fact that
             # f'(x) = (f(x+delta)-f(x-delta))/(2 delta) as delta -> 0
             gradient=(stress(add_delta(p,i,d,delta=0.001))-stress(subtract_delta(p,i,d,delta=0
             return gradient
```

```
In [ ]:
```

```
In [19]: def compute_full_gradient(p):
         # Numerically compute the full gradient of stress at a position p
         # This should return a matrix whose elements are the gradient of stress at p with res
             full_gradient=np.zeros((np.array(p).shape))
             for i in range(21):
                 for j in range(2):
                     full_gradient[i,j]=compute_gradient(p,i,j,delta=0.001)
             return full_gradient
```

```
In [20]: compute_full_gradient(pos)
```

```
365.2927557780574
365.3016039383682
365.2809488038164
365.31341845432473
365.2955737210888
365.2987857239974
365.27726666012404
```

365.31710358325415
365.27734630167345
365.31702470925677
365.30010329632387
365.2942540589594
365.28159635514385
365.31277158318477
365.28683370143926
365.30753128417734
365.2752846722249
365.31908331684343
365.31508917390056
365.27927700768987
365.29112843466487
365.3032234083214
365.29600075972115
365.2983582381439
365.283970205107
365.31039977568946
365.3012985984743
365.29305585013606
365.30232323829546
365.29204433556663
365.28345289482013
365.31091279938437
365.3596496817298
365.2347237893328
365.285294232108
365.30907280730355
365.3409940197782
365.2533792953586
365.3254782934502
365.2688907320643
365.29095369483787
365.30340748121904
365.2984720218516
365.29588410038565
365.2484040807417
365.34596840478474
365.3066048286971
365.28776012758175
365.29995970888365
365.2944031253311
365.31855766863976
365.2758125890581
365.30948355086275
365.2848838431451
365.25582964737237

```
365.33854267046354
365.3549027577362
365.23946989680724
365.2812013986836
365.3131651592879
365.3249579524344
365.26941371455155
365.27598318736295
365.3183784429319
365.27214542108896
365.32222221747406
365.2675041889654
365.3268657789571
365.30045234824775
365.29391392796595
365.31049354040044
365.2838714694702
365.24748856309066
365.34688294961404
365.3134249252254
365.2809406767307
365.30038309619357
365.2939850354944
365.3651530145552
365.22922243680534
365.29109586228003
365.3032589718543
365.29583986658724
365.29851243540907
```

```
Out[20]: array([[ -4.42408016, -16.23482525],
                [ -1.60600145, -19.91846157],
                [-19.83920379,   2.92461868],
                [-15.58761402, -10.34879137],
                [-21.89932231,  17.90608311],
                [ -6.04748683,  -1.17873921],
                [-13.21478529,   4.12137417],
                [  5.13945136, -13.72995228],
                [ 62.4629462 , -11.8892876 ],
                [ 43.80736221,  28.29378069],
                [ -6.22689319,   1.29396073],
                [-48.78216202,   9.42235056],
                [  2.77829178,  21.37253979],
                [ 12.29985386, -41.35651155],
                [ 57.71643046, -15.9818803 ],
                [ 27.77211894, -21.19762778],
                [-25.03839819, -29.680795  ],
```

```
        [  3.26921014,  13.31103547],
        [-49.69719326,  16.24212425],
        [  3.19903035,  67.96528887],
        [ -6.08155479,  -1.33628441]])
```

```
In [21]: # Now go through and adjust the position to minimize the stress
         pos# start positions
         rate=0.01 #learning rate
         max_iters=1000
         iters=0# iternation counter
         cur=pos
```

```
In [22]: while iters< max_iters:
             cur = cur- rate * compute_full_gradient(cur) #Grad descent
             iters = iters+1  #iteration count
             #print("Iteration",iters,"\nX value is",cur) #Print iterations

         #print("The local minimum occurs at", cur)
```

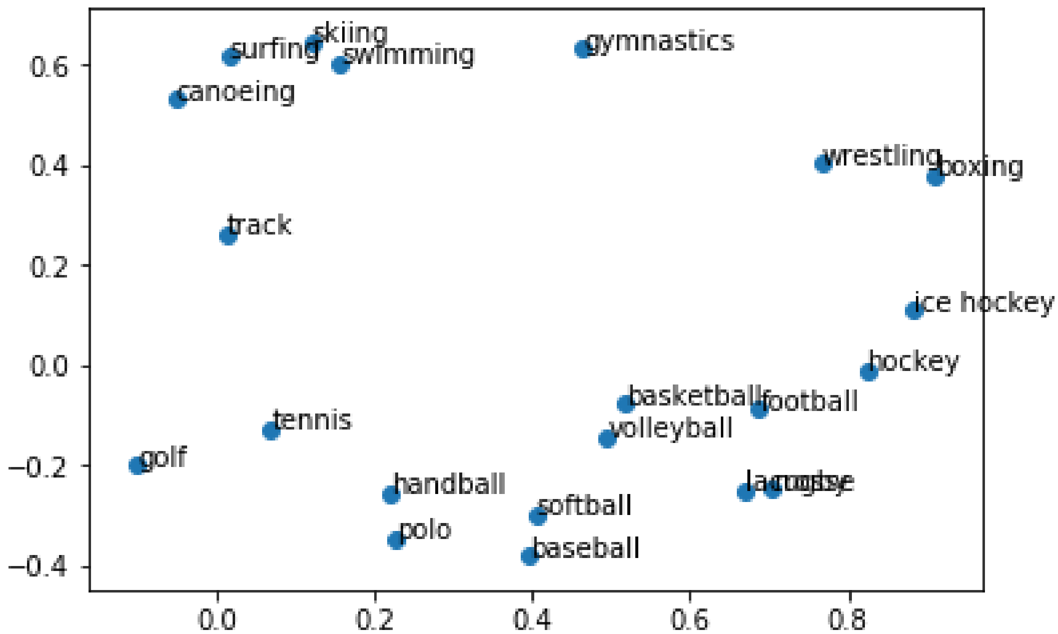
```
365.2927557780574
365.3016039383682
365.2809488038164
365.31341845432473
365.2955737210888
365.2987857239974
365.27726666012404
365.31710358325415
365.27734630167345
365.31702470925677
365.30010329632387
365.2942540589594
365.28159635514385
365.31277158318477
365.28683370143926
365.30753128417734
365.2752846722249
365.31908331684343
365.31508917390056
365.27927700768987
365.29112843466487
365.3032234083214
365.29600075972115
365.2983582381439
365.283970205107
365.31039977568946
365.3012985984743
365.29305585013606
365.30232323829546
```

```
9.389991372614626
9.389983920372261
9.38998392058927
9.389984852392093
9.389984852205798
9.389988231911722
9.389988231986443
9.389986847236099
9.389986847466215
9.389989759863276
9.389989760029493
9.389986597575582
9.389986597713953
9.389987915369044
9.389987915209414
9.389985781812012
9.389985782048797
9.389988738480868
9.389988738452688
```

```
In [23]: #plot example with annotation
         cur=np.transpose(cur)
         x=cur[0]
         y=cur[1]
         n=names

         fig, ax = plt.subplots()
         ax.scatter(x,y)


         for i, txt in enumerate(names):
             ax.annotate(txt, (x[i], y[i]))
```




```
In [24]: #Do the result agree with expectation?
         #Yes, the result pretty much meet the expectation with my psychological distance. Sin
         #in one cluster, boxing and Wrestling are indeed th most similar.
```

```
In [25]: #3. [5pts] Plot the stress over iterations of your MDS. How should you use this plot
         #how many iterations are needed?
         #I might need 2000-3000 iterations to capture the local minimum. The plot would be a
         #I decided to select the cut off when there are 2000 iterations.
```

```
In [28]: #4. [10pts] Run the MDS code you wrote 5 times and show small plots, starting from ra
         #positions. Are they all the same or not? Why?
         #No, they are not the same. Since gradient descent would help us find local minimum i
         #might converge at different local minimms.
         def mds():
             pos# start positions
             rate=0.01 #learning rate
             max_iters=1000
             iters=0# iternation counter
             cur=pos

             while iters< max_iters:
                 cur = cur- rate * compute_full_gradient(cur) #Grad descent
                 iters = iters+1  #iteration count
               # print("Iteration",iters,"\nX value is",cur) #Print iterations
```

```
cur=np.transpose(cur)
x=cur[0]
y=cur[1]
n=names

fig, ax = plt.subplots()
ax.scatter(x,y)


for i, txt in enumerate(names):
    ax.annotate(txt, (x[i], y[i]))
```

In [29]:
```
#5.If you wanted to find one best answer but had run MDS 5 times, how would you pick
#best? Why?
#I would pick the minimum of the stress out of all stresses generated. This would the
#plotted would be the closest to reported similarites.
```