```
import numpy as np
import math
import matplotlib.pyplot as plt
import datetime
import random

"""Problem 1 COMPLETE
    The dataset1 provides similarities, not distances. Write down three
    ways you could convert a similarity to a distance ψij and choose one to use in the
    code. Briefly explain why you chose it over the others.
    1. Use the formula distance = (1 - similarity)
    2. Use the formula distance = 1 / similarity
    3. User the formula distance = 1 / (similarity)**2

    I will use distance = (1 - similarity) because it maintains the same scale as
    the similarity matrix and doesn't require normalizing. It is also easier for me
    to conceptualize, if two categories have a similarity score of 1
    then their distance = 1 - 1 = 0 meaning they are in the same location. """

def simToDist(x):
    """Converts a similarity to a psychological distance"""
    return 1 - x
"""END PROBLEM 1"""

"""Problem 2 COMPLETE
    Write a function that takes a vector/matrix of positions for each item and computes
        the stress."""

def stress(psychArray, coordArray):
    """Returns the stress value of the current graph"""
    stressSum = 0

    for i in range(len(coordArray) - 1):
        for j in range(i + 1, len(coordArray)):
            target = coordArray[i]
            dest = coordArray[j]
            stressSum += (psychArray[i, j] - np.linalg.norm(target - dest)) ** 2

    return stressSum

"""End Problem 2"""

"""Problem 3, COMPLETE
    Write down a function that takes a vector/matrix of positions and computes the
gradient
    (e.g. applies the above numerical method of df/dp to each coordinate location)."""
```

```
def gradient(point, psychArray, coordArray, h=0.001):
    """Returns the gradient using partial derivative method with respect to x and y in a
single point"""
    plusArrayX = np.copy(coordArray)
    minusArrayX = np.copy(coordArray)
    plusArrayY = np.copy(coordArray)
    minusArrayY = np.copy(coordArray)
    minusArrayX[point][0] -= h
    plusArrayX[point][0] += h
    minusArrayY[point][1] -= h
    plusArrayY[point][1] += h

    dx = (stress(psychArray, plusArrayX) - stress(psychArray, minusArrayX)) / (2*h)
    dy = (stress(psychArray, plusArrayY) - stress(psychArray, minusArrayY)) / (2*h)

    return dx, dy
"""END PROBLEM 3"""

"""Problem 4 COMPLETE
Write the code that follows a gradient in order to find positions that minimize the stress –
be sure to
take small steps in the direction of the gradient (e.g. 0.01*gradient). Plot the sport
names at the resulting
coordinates. Do the results agree with your intuitions about how this domain might be
organized? Why or why not?

    ANSWER: In general I think the sports are grouped appropriately although there are
some exceptions.
    The water related sports swimming, canoeing, skiing, and surfing are all nicely
grouped. Ball related
    sports are all in the same area of the grid although the distances between them aren't
always
    consistent with the psychological distances. Because this is only one run of the MDS,
on one random
    set of points, the MDS may have found a local minimum rather than a global
minimum. Similar sports may be
    in the same area of the grid but not as close as they could be if the global minimum
was found.  """

def traceGradient(psycho_array, learn_rate=.01, gradient_threshold=0.0005, n=1000,
dimensions=2):
    """Takes a psychological distance array, returns a position array after n iterations, it's
stress value,
        and the stress values over iterations as stressList"""

    grad_x, grad_y = 10000, 10000
    grad_total = 10000
```

```
    min_stress = float('inf')
    stress_value = float('inf')
    best_positions = None
    stressList = []
    bestIter = 0
    position_array = getRandPositions(psycho_array.shape[0], dimensions)
    z = 0
    x, y = 0, 1

    while (z < n):

        for point in range(0, len(position_array)):
            grad_x, grad_y = gradient(point, psycho_array, position_array, h=.001)
            position_array[point][x] += (-grad_x * learn_rate)
            position_array[point][y] += (-grad_y * learn_rate)

        stress_value = stress(psycho_array, position_array)
        stressList += [stress_value]

        z += 1
        if z % 100 == 0:
            print(z)

    return position_array, stress_value, stressList, bestIter

def importCSV(csvFile, dataType, header=1):
    """Returns a numpy array without headers from a CSV file"""
    if header:
        csvArray = np.genfromtxt(csvFile, dtype=dataType ,delimiter=",", skip_header=1)
    else:
        csvArray = np.genfromtxt(csvFile, dtype=dataType ,delimiter=",")
    return csvArray

def convertArray(simArray, func):
    """Takes a similarity array and converts it to a psychological distance array using
        the supplied function func"""

    return np.apply_along_axis(func, 1, simArray)

def getRandPositions(rows, columns):
    """Returns a rows by columns matrix with random values"""
    return np.random.randn(rows, columns)

def getNames(csvFile, dataType=str, header=1):
    """Returns the header names form a CSV file"""
    array = np.genfromtxt(csvFile, dtype=str, delimiter=",", names=True)
    return list(array.dtype.names)
```

```
def makeLabels(nameList, positionArray, font_size=5):
    fig, labels = plt.subplots()

    for j in range(len(positionArray)):
        labels.annotate(nameList[j], positionArray[j], size=font_size, ha='left')

def saveMdsArrays(psycho_array, k=10, trials=1000, dim=2):
    """Saves MDS arrays as csv files"""
    stressTrace = []
    for z in range(1, k + 1):
        dataArray, stressVal, stressTrace, bestIteration = traceGradient(psyArray,
learn_rate=.01, n=trials, dimensions=dim)
        name = 'MDS_n_1000_UPDATE_' + str(z) + '.csv'
        np.savetxt(name, dataArray, delimiter=',')

        stressName = 'Stress_n_1000_UPDATE_' + str(z) + '.csv'
        np.savetxt(stressName, stressTrace, delimiter=',')

#driver code for questions 1-4
sportData = 'similarities.csv'
circleData = 'circle.csv' #to test known shape

importArray = importCSV(sportData, float)
nameArray = getNames(sportData, float)

psyArray = convertArray(importArray, simToDist)
arraySize = psyArray.shape[0]
dim = 2

############################## SAVE BELOW CODE
# x2, y2 = zip(*posArray_mod)
# colors1 = np.random.RandomState(0).rand(arraySize)
# # makeLabels(nameArray, posArray_mod, 5)
# plt.title("MDS For Psychological Similarity Distances of Sports, n=1000")
# plt.scatter(x2,y2, c=colors1)
# # plt.savefig('n1000_newMethod_1.pdf') #change name of file for future saves
# plt.show()
# plt.close()
############################## SAVE ABOVE CODE
# saveMdsArrays(psyArray, k=9) #remove after saving
"""END PROBLEM 4"""


"""Problem 5 COMPLETE
    Make a scatter plot of the the pairwise distances MDS found vs. people’s reported
distances.
```

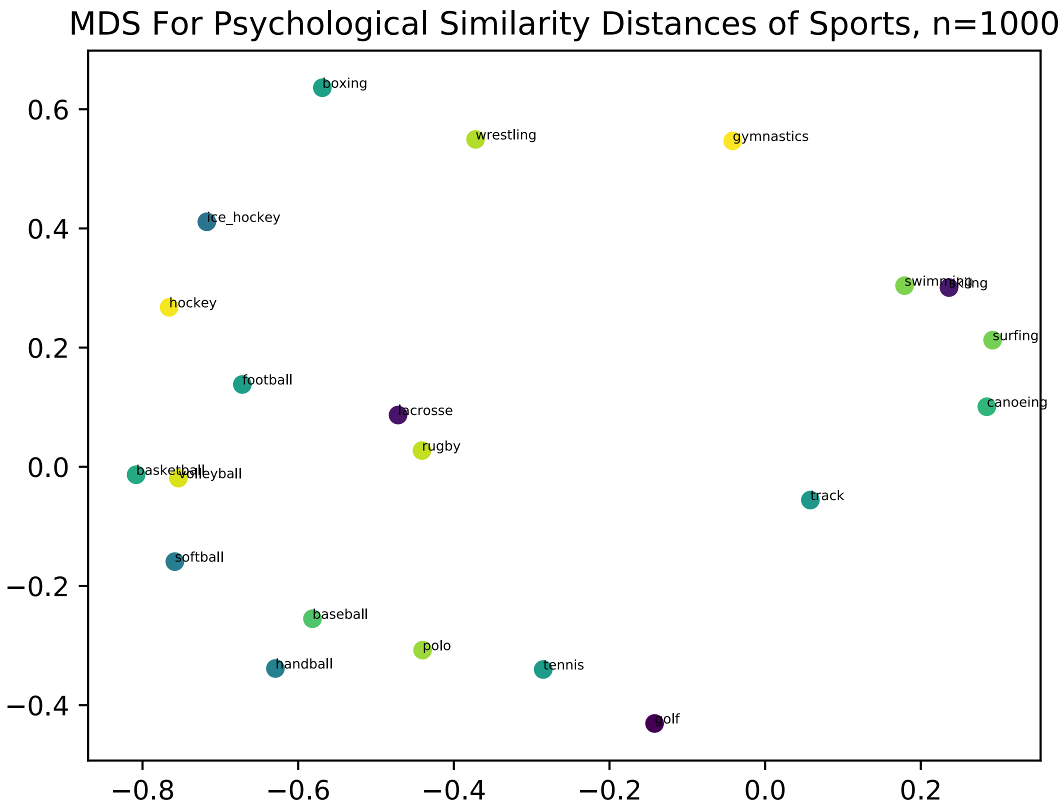
```
    Briefly describe what good and bad plots would look like and whether yours is good
or bad.

    ANSWER: I plotted x= MDS distances for each pair, y=reported distances for each
pair of sports. I would
    expect that, for a perfect fit, all of the dots would cluster closely with the blue line
y=x.
    My plot doesn't look great but then I don't know what an optimal version of this MDS
algorigthm
    with this data would look like. There are more points above y=x than below meaning
that most of
    my MDS distances are closer together than they should be according to the similarity
distances.
    """

def distancesMDS(mdsDist, psychArray):
    psyPlot = psychArray.flatten()
    lineListX, lineListY = [0], [0]
    x,y = 0, 0

    mdsList = []

    for i in range(len(mdsDist)):
        for j in range(len(mdsDist)):
            mdsList += [np.linalg.norm(mdsDist[i] - mdsDist[j])]
            x += .0024
            y += .0024
            lineListX += [x]
            lineListY += [y]

    #normalize MDS distances to match bounds of psychological distances
    normMDS = max(mdsList)
    mdsList = [(z / normMDS) for z in mdsList]

    plt.scatter(mdsList, psyPlot, c='red')
    plt.plot(lineListX, lineListY, c='blue')
    plt.title('MDS Pairwise Distances vs. Reported Similarity Distances (Soph)')
    plt.xlabel('MDS Distances')
    plt.ylabel('Similarity Distances')
    # plt.savefig('P5_pairwise_naive.pdf')
    # plt.savefig('P5_pairwise_soph.pdf')
    plt.show()
    plt.close()

mdsDist1 = importCSV('MDS_n_1000_UPDATE_10.csv', float, header=0)
distancesMDS(mdsDist1, psyArray)
# mdsDist2 = importCSV('MDS_n_1000_1.csv', float, header=0)
```

```
# distancesMDS(mdsDist2, psyArray)

"""Problem 6 COMPLETE
    Plot the stress over iterations of your MDS. How should you use this plot in order
    to figure out how many iterations are needed?

    ANSWER: The stress levels out and does not change after n iterations. The number
of
    iterations at the point where stress no longer changes is how many are needed.
    This particular graph declines very sharply early on indicated that it may have
reached
    a local minimum rather than a global minimum."""

def plotStress(stressCSV, iterations=1000):
    stressData = importCSV(stressCSV, float, header=0)
    stress_minima = min(stressData)
    pY = stressData
    pX = list(range(1, iterations + 1))
    poly = np.polyfit(pX, pY, 4) #graph is not showing the minimum accurately, redo
    pYpoly = np.poly1d(poly)(pX)

    plt.plot(pX, pY, c='red', label='stress minima = ' + str(int(stress_minima)))
    plt.scatter(106, stress_minima, c='blue', label='stress minima')
    plt.title("Stress Over Iterations")
    plt.xlabel('Iterations')
    plt.ylabel('Stress')
    plt.legend()
    # plt.savefig('stress_over_iter_1000_5.pdf')
    plt.show()
    plt.close()

plotStress('Stress_n_1000_UPDATE_5.csv')

"""END PROBLEM 6"""

"""Problem 7 COMPLETE
    Run the MDS code you wrote 10 times and show small plots, starting from random
initial
    positions. Are they all the same or not? Why?

    ANSWER: The plots are not all the same. First, they are all starting from different
    random positions and moving towards their ideal positions using MDS so groupings
will
    be in different areas of the graph. Second, the MDS is only being run one time on
each
    graph so they may only be achieving a local minima and missing the global minima.
Ideally
```

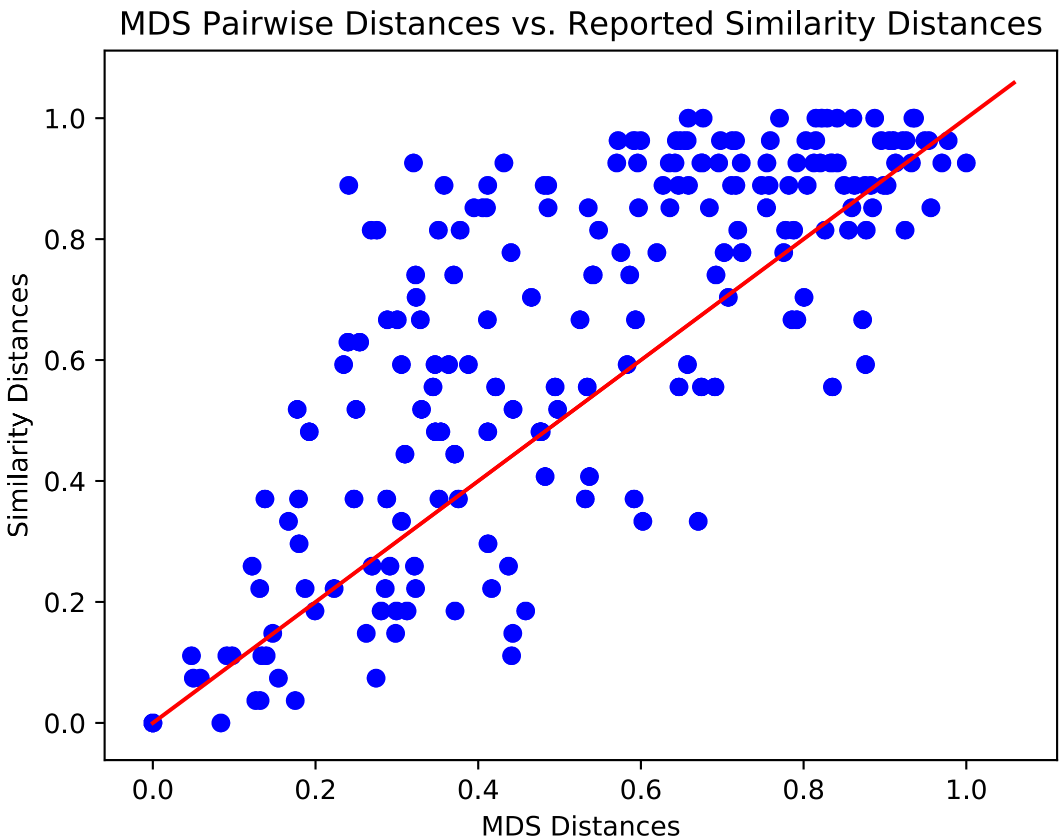
```
    the MDS would run until a global minima was reached. The only way to do this is to
create
    a new random array of coordinates, move on the gradient until it is near 0, and
record
    the stress. Over a sufficient number of iterations we should expect that the array with
    the lowest stress has reached the global minimum and can then see the best fit. """

def plotMDS(*csvFiles, names, rows=2, cols=5):
    pointNames = getNames(names, str)
    # print(pointNames)
    row, col = 2, 5
    fig, mds = plt.subplots(row, col)
    data = []
    t = 0
    scatterColor = np.random.RandomState(0).rand(21)
    # print(scatterColor)

    for csv in csvFiles:
        data += [importCSV(csv, float, header=0)]

    for i in range(row):
        for j in range(col):
            xVal, yVal = zip(*data[t])
            color = scatterColor[t]
            mds[i, j].scatter(xVal, yVal, c=scatterColor)
            header = 'subplot ' + str(t + 1)
            mds[i, j].set_title(header)
            for k, txt in enumerate(pointNames):
                mds[i, j].annotate(txt, (xVal[k], yVal[k]), size=5)
            t += 1

    fig.set_figheight(10)
    fig.set_figwidth(15)
    fig.suptitle('MDS Plots N=1000', fontsize=16)
    plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
    plt.savefig('Q7_10_UPDATE_plots.pdf')
    plt.show()

plotMDS('MDS_n_1000_UPDATE_1.csv',
'MDS_n_1000_UPDATE_2.csv','MDS_n_1000_UPDATE_3.csv','MDS_n_1000_UPDAT
E_4.csv','MDS_n_1000_UPDATE_5.csv','MDS_n_1000_UPDATE_6.csv',

'MDS_n_1000_UPDATE_7.csv','MDS_n_1000_UPDATE_8.csv','MDS_n_1000_UPDAT
E_9.csv','MDS_n_1000_UPDATE_10.csv', names=sportData)

"""END PROBLEM 7"""
```

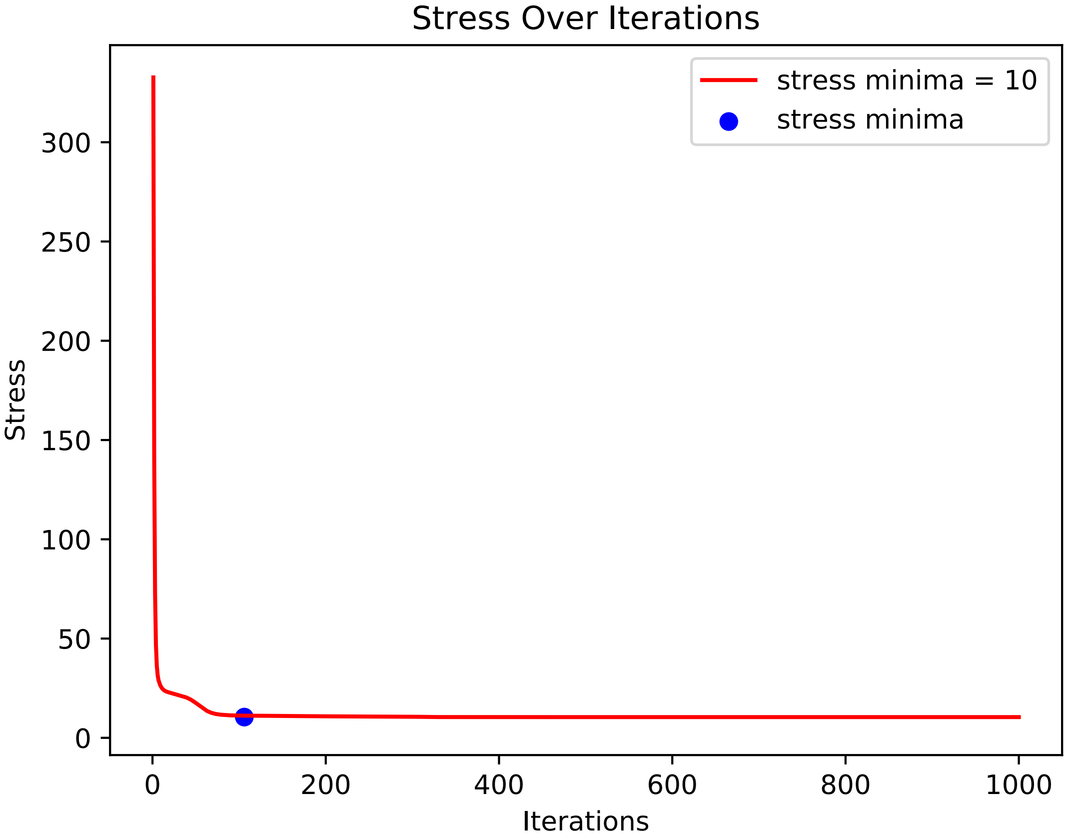
```
"""Problem 8 COMPLETE
    If you wanted to find one “best” answer but had run MDS 10 times, how would you
pick
    the best? Why? Show a plot of the best and any code you used to find it.

    ANSWER: The 'best' answer would be the MDS array that achieved the lowest stress.
There is
    accurate way to choose the best MDS simply by looking at the 10 subplots. To find
the best,
    I changed my traceGradient algorigthm (actually it's the first one I used as it's more
accurate
    and I didn't realize the assignment wanted a less accurate version) to do the
following:

    1) Create a new random array of x,y points.
    2) Adjust the points using the gradient until a threshold near 0 is reached.
    3) Check the stress for the point array and save the array if it's stress score is less
than
        the last best point array.
    4) Repeat 1-4 n iterations.
    5) Return the best array and plot.

    This method will help ensure that a global minimum is found because we increase the
chances
    that we get a random point array that is near the global minimum which should result
in the
    lowest possible stress.

    I have included a plot using this method which shows a better clustering of sports
than
    doing N iterations on one position array. """

def traceGradientOptimal(psycho_array, learn_rate=.01, gradient_threshold=0.0005,
n=1000, dimensions=2):
    """Takes a psychological distance array, returns a position array with min(stress) after
N iterations,
        it's stress value, and the total stress for each position array over N iterations"""

    grad_x, grad_y = 10000, 10000
    grad_total = 10000
    min_stress = float('inf')
    stress_value = float('inf')
    best_positions = None
    stressList = []
    bestIter = 0

    for i in range(n):
```

```
        x, y = 0, 1

        position_array = getRandPositions(psycho_array.shape[0], dimensions)
        while (grad_x > gradient_threshold) and (grad_y > gradient_threshold):
            for point in range(0, len(position_array)):
                grad_x, grad_y = gradient(point, psycho_array, position_array, h=.001)
                position_array[point][x] += (-grad_x * learn_rate)
                position_array[point][y] += (-grad_y * learn_rate)

        grad_x, grad_y = 10000, 10000

        stress_value = stress(psycho_array, position_array)
        stressList += [stress_value]

        if stress_value < min_stress:
            min_stress = stress_value
            best_positions = position_array
            bestIter = i

    return best_positions, min_stress, stressList, bestIter
"""END PROBLEM 8"""
#end
```



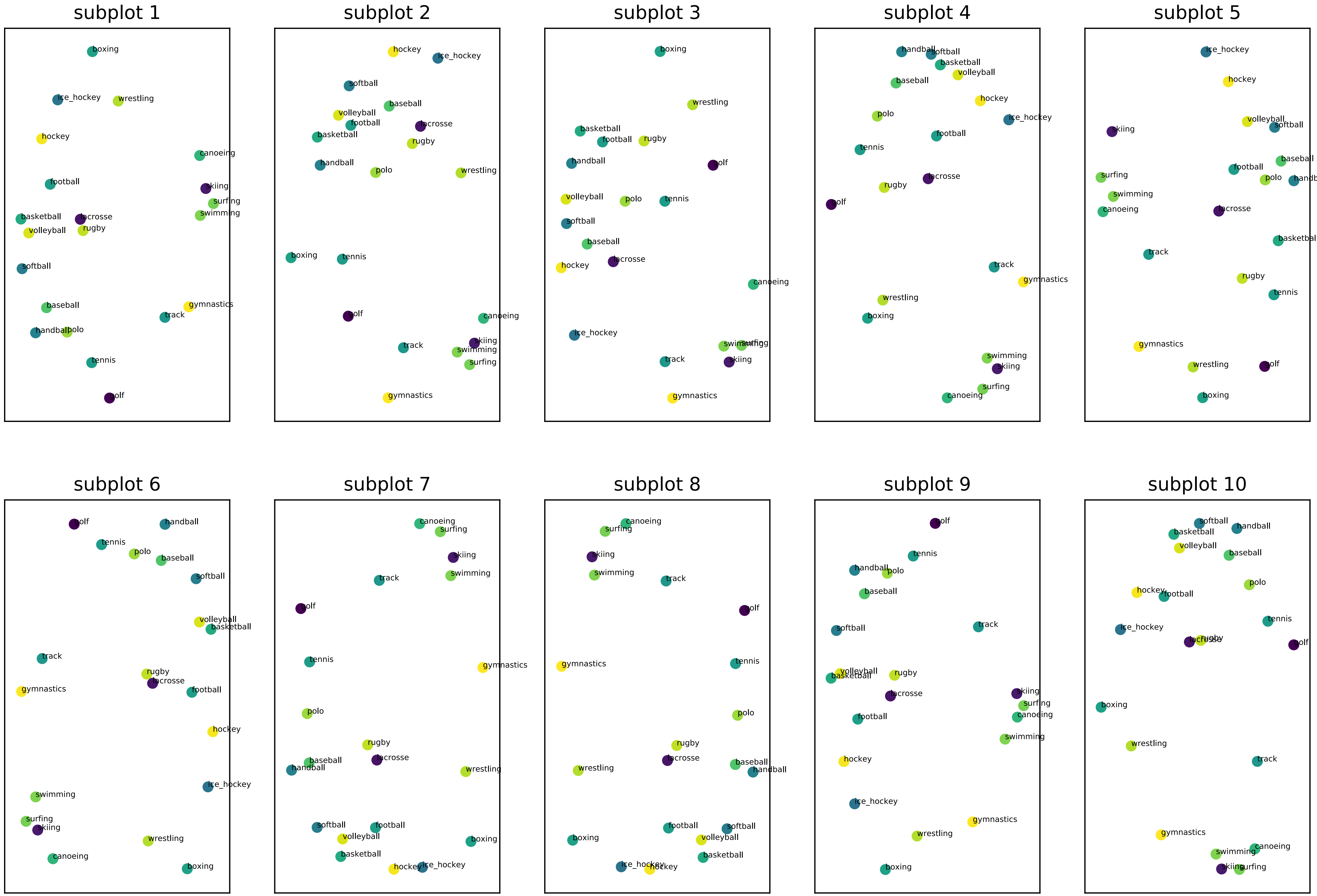
MDS For Psychological Similarity Distances of Sports, n=1000
boxing
wrestling
gymnastics
ice_hockey
swimming
skiing
hockey
surfing
football
canoeing
lacrosse
rugby
basketball
volleyball
track
softball
baseball
polo
handball
tennis
golf
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.8
−0.6
−0.4
−0.2
0.0
0.2

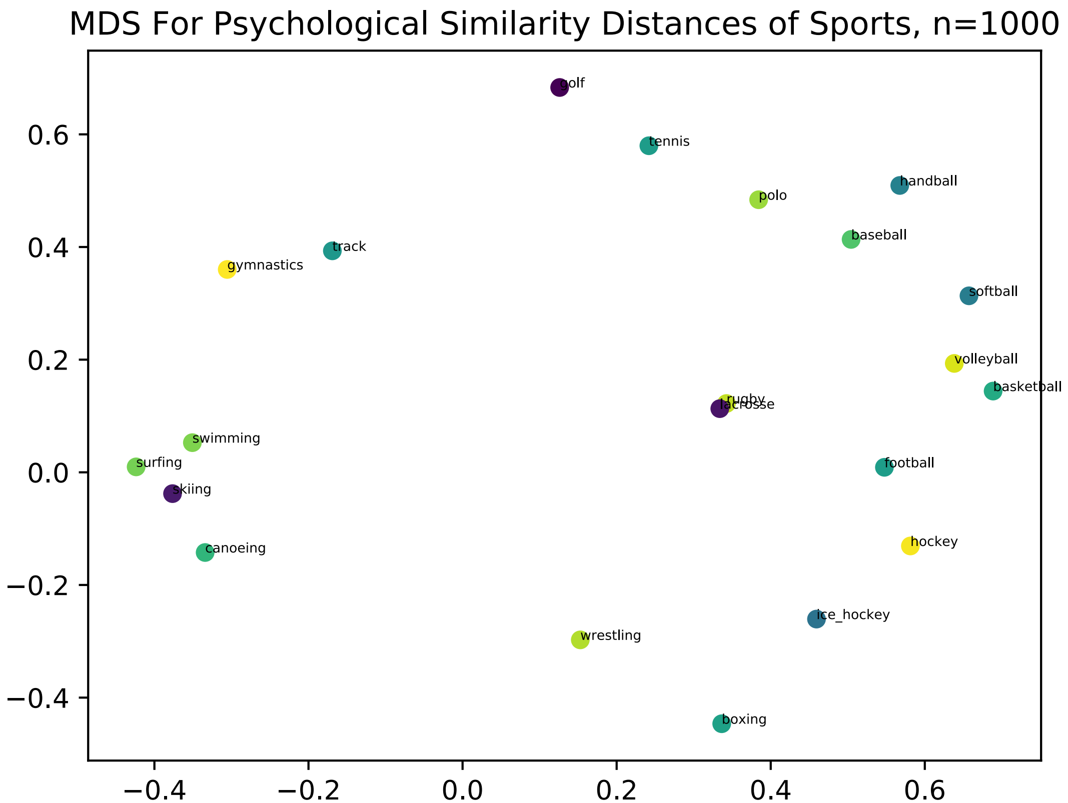

MDS Pairwise Distances vs. Reported Similarity Distances
Similarity Distances
MDS Distances
0.0
0.2
0.4
0.6
0.8
1.0



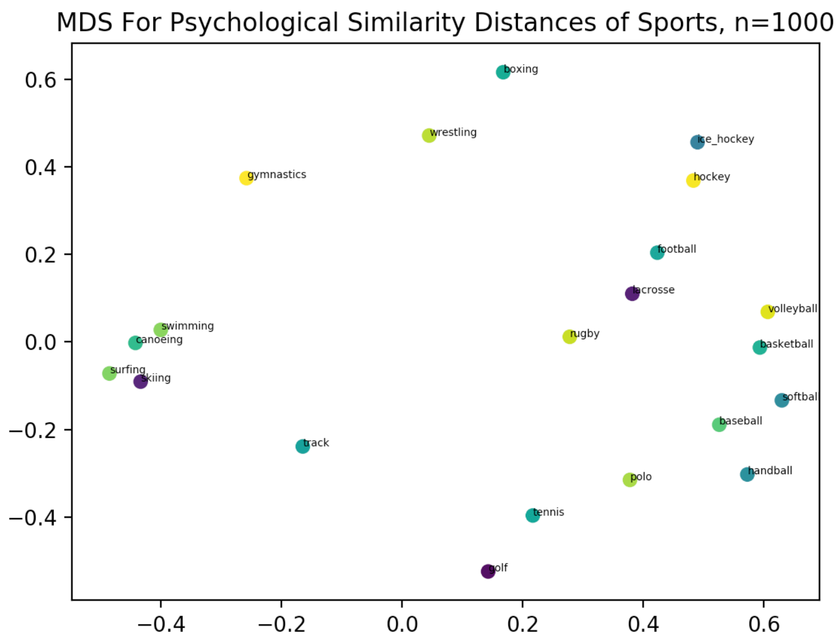
Stress Over Iterations
stress minima = 10
stress minima
Stress
Iterations
0
50
100
150
200
250
300
0
200
400
600
800
1000

MDS Plots N=1000

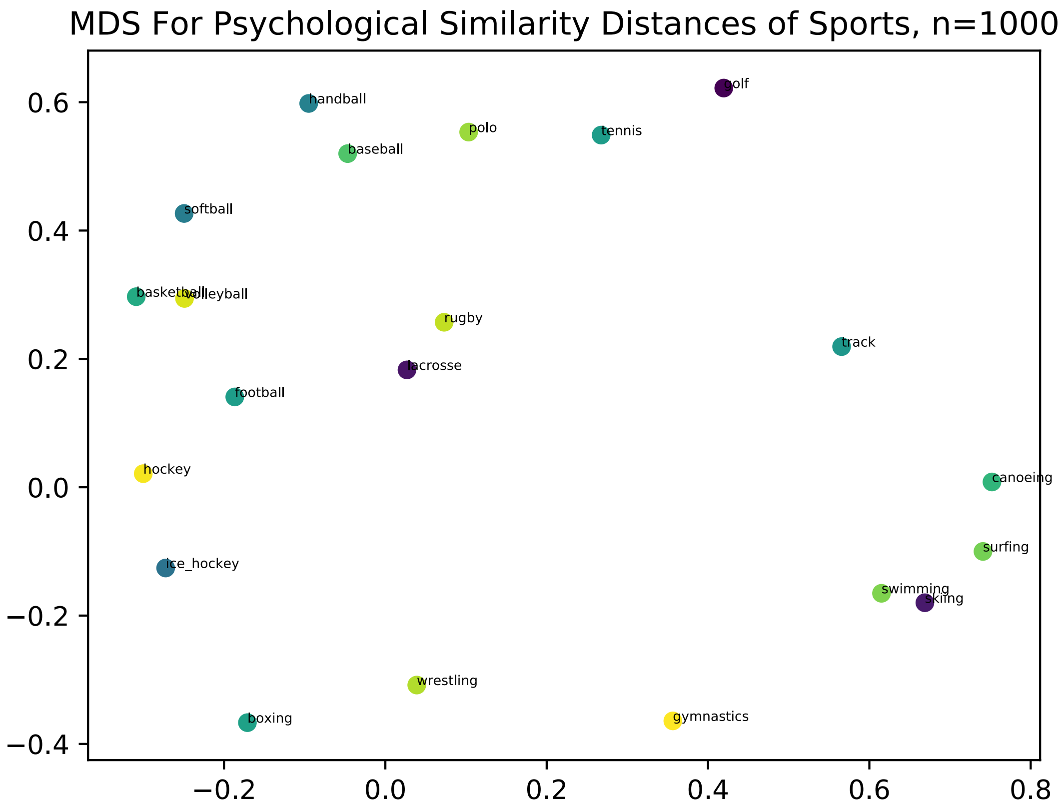




MDS For Psychological Similarity Distances of Sports, n=1000
golf
tennis
handball
polo
baseball
track
gymnastics
softball
volleyball
basketball
rugby
lacrosse
swimming
surfing
football
skiing
canoeing
hockey
ice_hockey
wrestling
boxing
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6



MDS For Psychological Similarity Distances of Sports, n=1000
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6
boxing
wrestling
ice_hockey
gymnastics
hockey
football
lacrosse
volleyball
swimming
canoeing
rugby
basketball
surfing
skiing
softball
baseball
track
handball
polo
tennis
golf



MDS For Psychological Similarity Distances of Sports, n=1000
handball
golf
polo
tennis
baseball
softball
basketball
volleyball
rugby
track
lacrosse
football
hockey
canoeing
surfing
ice_hockey
swimming
skiing
wrestling
boxing
gymnastics
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8