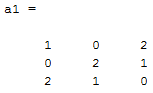
**Design of experiment and data analysis Tutorial 3 Class work**

**Sample calculations in t-SNE**

**Computation step 1: Calculation of distance matrix in high dimension:**

**Paper and pencil exercise:** Calculate the distance matrix and show the calculation in details for one calculation. What is the dimension of the distance matrix?

******

**Calculate the distance matrix:**

**Python Code:**

from scipy.spatial.distance import cdist

#D1 is the high dimensional numpy array

# Calculate distance matrix using Euclidean distance

dist\_matrix = cdist(D1, D1, metric='euclidean')

print(dist\_matrix)

**Computation step 2: Calculation of similarity (joint probability) in High dimension: [P(i,j)]** Suppose the following is the matrix for 6 points for high dimension data (n dimension). Calculate the conditional probability and then joint probability matrix/similarity matrix. With **sigma =1.00**

**Code hints for the High dimensional matrix are shared, try to replicate it with low dimensional matrix.**

**## Import the libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.spatial.distance import cdist

**## Load the data**

data3 = [[0.000, 2.000, 1.998, 1.181, 1.812, 1.993],

        [2.000, 0.000, 1.999, 1.216, 1.795, 1.996],

        [1.998, 1.999, 0.000, 1.258, 1.772, 1.998],

        [1.181, 1.216, 1.258, 0.000, 0.208, 1.326],

        [1.812, 1.795, 1.772, 0.208, 0.000, 1.732],

        [1.993, 1.996, 1.998, 1.326, 1.732, 0.000]]

data3 = np.array(data3)

sns.heatmap(data3)

**## Calculate the distance matrix**

dist\_matrix3 = cdist(data3, data3, metric='euclidean')

A math equation with black text

Description automatically generated## **Calculate the conditional probability**

def sim(dist):

    return math.exp(-(dist)\*\*2/2)

def cond\_prob(j,i,dist\_mat):

    # i and j are shapes of the matrix, in python indexing starts from 0 so we need to subtract 1

    i = i-1

    j = j-1

    # We are calculating the numerator for the conditional probability

    num = sim(dist\_mat[i,j])

    # We are calculating the denominator for the conditional probability

    if i ==j:

        return 0 # Refer the formula

    else:

        den = 0

        for col in range(np.shape(dist\_mat)[1]): # We are iterating over the columns

            if col != i:

                den += sim(dist\_mat[i,col])

        return num / den # Returning the conditional probability

print('The conditional probability of i=1 and j=3 is: ',cond\_prob(3,1,dist\_matrix3))

print('The conditional probability of i=3 and j=1 is: ',cond\_prob(1,3,dist\_matrix3))

# Check are the probabilities different or same, refer literature to find out if you are correct?

**## Calculate the joint probability:**

**A math equation with black text

Description automatically generated**

def joint\_prob(dist\_matrix,data):

    # Our conditional matrix will be of same shape as of the distance matrix

    cond\_matrix = np.zeros(np.shape(dist\_matrix))

    row, colm = np.shape(data)

    m,n = np.shape(dist\_matrix)

**A math equation with black text

Description automatically generated**    # We are iterating over the rows and columns to find out the conditional probability for each pair

    for i in range(1,m+1):

        for j in range(1,n+1):

            cond1 = cond\_prob(j,i,dist\_matrix)

            cond2 = cond\_prob(i,j,dist\_matrix)

            # Refer the formula

            temp = (cond1 + cond2)/(2\*row\*colm)

            # Storing the value in the conditional matrix

            cond\_matrix[i-1,j-1] = temp

    return cond\_matrix

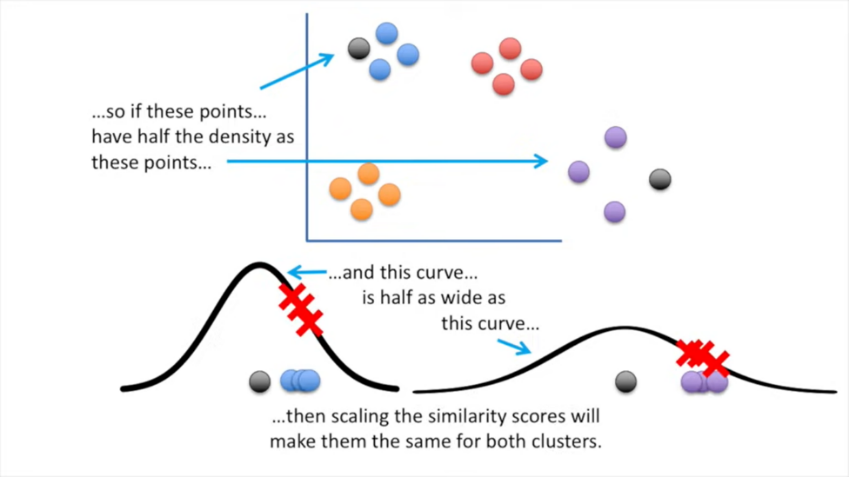
joint\_prob\_data3 = joint\_prob(dist\_matrix3,data3)

print('The high dimensional Matrix is: ')

print(np.round(data3,3))

print(data3.shape)

print('\nThe joint probability of high dimension matrix is: \n\n',np.round(jp\_hig\_dim,4))



**Now we are assuming we have a reduced dimension matrix of our data, we will initialize a random matrix of 10 data points and 2 dimensions. We will later make this matrix as close as possible to the high dimension matrix by using Gradient Descent.**

**Initialization of 2 dimensional matrix:**

#Calculation of joint probability for low dimension matrix

#############################

#Code to generate the random number matrix

np.random.seed(0)

Y = np.random.rand(6,2)

print('The high dimensional Matrix is: ')

print(np.round(data3,3))

print(data3.shape)

print('\n\nThe low dimensional Matrix is: ')

print(np.round(Y,5))

print(Y.shape)

**Do it yourself:**

**Computation step 4: Calculation of distance matrix of the above low dimension matrix (guess value)**

**Computation step 5: Calculation of the conditional probability, and similarity (joint probability) in low dimension from the above distance matrix.**

**A black and white math symbol

Description automatically generatedComputation step 6: Calculate the KL divergence betwen the joint probabilities for the first iteration.**

def kl\_divergence(P, Q):

    """Compute the KL divergence between joint probability matrices P and Q."""

    epsilon = 1e-8  # Small value to avoid division by zero

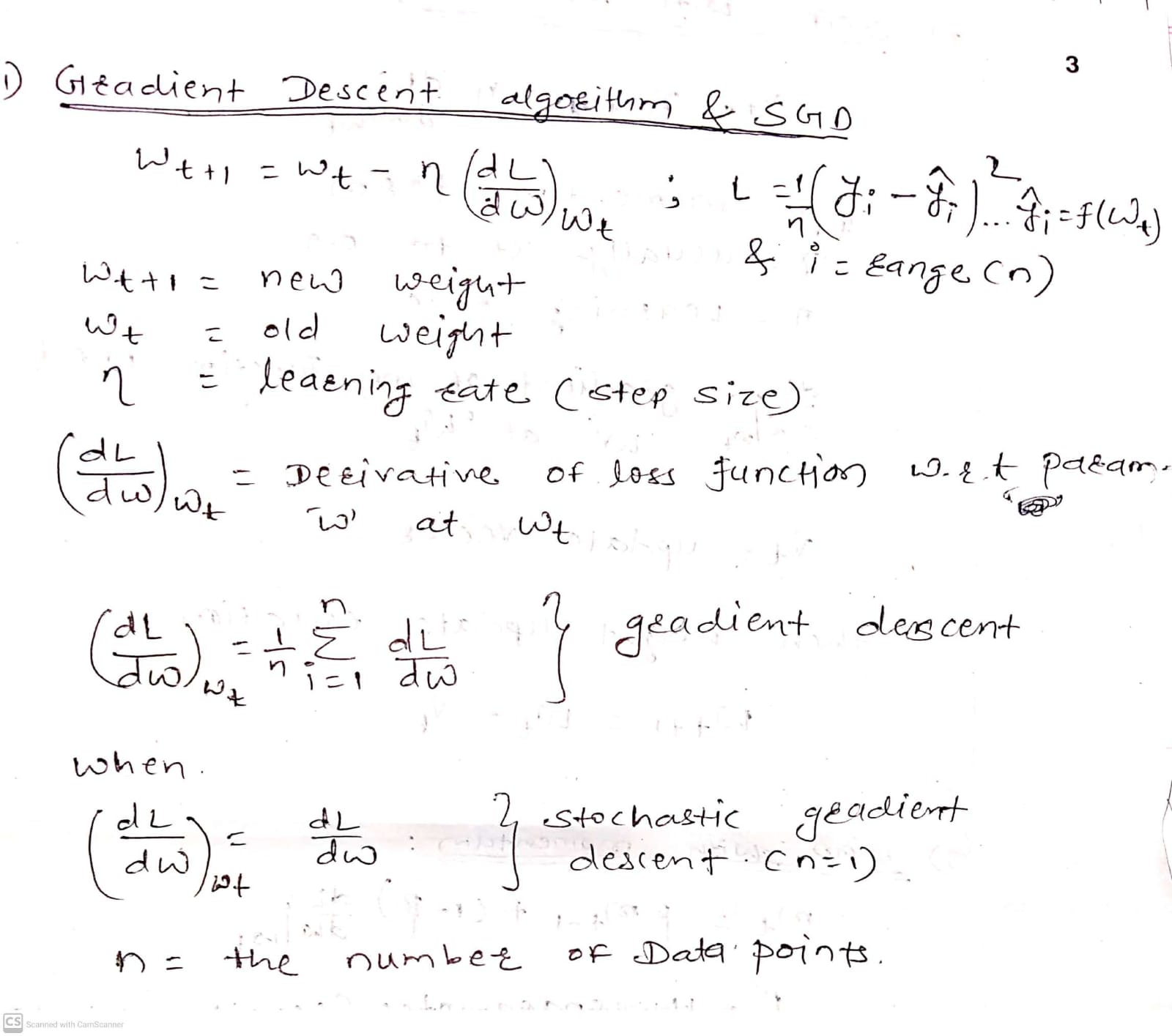
    P = np.clip(P, epsilon, None)

    Q = np.clip(Q, epsilon, None)

    kl\_div = np.sum(P \* np.log(P / Q))

    return kl\_div

**Computation step 7: Set a code for gradient descent algorithm to minimize the KL divergence between P and Q, the joint probability of high and low dimension matrix:**



**Code Hints:**

**Minimize KL Div. loss for two normal distributions. From this code you will learn how to use gradient descent in pytorch, and use it on a matrix. Here your parameter which is being optimized is a matrix.**

**(Modify the code to apply for the given matrices in TSNE)**

# Learning to optimize the KL divergence of two probability distributions one is known and another is randomly generated

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import norm

#from scipy.stats import entropy as kl\_divergence

def kl\_divergence(p, q):

    return np.sum(np.where(p != 0, p \* np.log(p / q), 0))

x = np.arange(-10, 10, 0.01)

#mean = 0, standard deviation = 2

p\_pdf = norm.pdf(x, 0, 2 )

#Generate a random mean, random standard deviation and random pdf

np.random.seed(32)

random\_mean = np.random.randint(10)

random\_std = np.random.randint(10)

q\_pdf = norm.pdf(x, random\_mean, random\_std)

plt.plot(x, p\_pdf, label='p(x)')

plt.plot(x, q\_pdf, label='q(x)')

plt.legend()

plt.show()

print('The KL Divergence between the two distributions is: ',kl\_divergence(p\_pdf, q\_pdf))

import torch

learning\_rate = 0.001

epochs = 1000

# In pytorch we need to define the variables which will be optimzed after gradient calculation

# Formula for normal distribution is given by: 1/(std\*sqrt(2\*pi)) \* exp(-0.5\*((x-mean)/std)\*\*2)

x = torch.arange(-10, 10, 0.01, dtype=torch.float)

p = torch.tensor(p\_pdf, dtype = torch.float)

mu = torch.tensor(4, requires\_grad=True, dtype = torch.float)

std = torch.tensor(6, requires\_grad=True, dtype = torch.float)

#Mean is initialized with 4 and standard deviation is initialized with 6

#Define Optimizer

optimizer = torch.optim.SGD([mu, std], lr = learning\_rate)

epoch\_count = []

loss\_count = []

mean = []

variance = []

for epoch in range(epochs):

    q\_pdf = torch.exp(-torch.square(torch.tensor(x) - mu) / (2 \* std\*\*2)).type(torch.float)

    q = q\_pdf / torch.sum(q\_pdf)

    kl\_div = torch.sum(p \* torch.log(p / q))

    optimizer.zero\_grad()

    kl\_div.backward()

    optimizer.step()

    epoch\_count.append(epoch)

    loss\_count.append(kl\_div.item())

    mean.append(mu.item())

    variance.append(std.item())

plt.figure(figsize = (12, 4))

plt.subplot(1, 3, 1)

plt.plot(epoch\_count, loss\_count)

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('KL Divergence Loss')

plt.subplot(1, 3, 2)

plt.plot(mean, label=f'Current Mean: {np.round(mu.item(),2)}')

plt.axhline(y=0, color='r', linestyle='--', label='Actual: 0')

plt.title('Mean, Actual Mean: 0')

plt.legend()

plt.subplot(1, 3, 3)

plt.plot(variance, label=f'Current Var: {np.round(std.item(),2)}')

plt.axhline(y=2, color='r', linestyle='--', label='Actual: 2')

plt.title('Variance')

plt.legend()

plt.show()

**A graph of a graph

Description automatically generated with medium confidence**