def perceptronStep(X, y, W, b, learn\_rate = 0.01):

for i in range(len(X)):

y\_hat = prediction(X[i],W,b)

if y[i]-y\_hat == 1:

W[0] += X[i][0]\*learn\_rate

W[1] += X[i][1]\*learn\_rate

b += learn\_rate

elif y[i]-y\_hat == -1:

W[0] -= X[i][0]\*learn\_rate

W[1] -= X[i][1]\*learn\_rate

b -= learn\_rate

return W, b

0.12419,0.33595,1

0.25644,0.42624,1

0.4591,0.40426,1

0.44547,0.45117,1

0.42218,0.20118,1

0.49563,0.21445,1

0.30848,0.24306,1

0.39707,0.44438,1

0.32945,0.39217,1

0.40739,0.40271,1

0.3106,0.50702,1

0.49638,0.45384,1

0.10073,0.32053,1

0.69907,0.37307,1

0.29767,0.69648,1

0.15099,0.57341,1

0.16427,0.27759,1

0.33259,0.055964,1

0.53741,0.28637,1

0.19503,0.36879,1

0.40278,0.035148,1

0.21296,0.55169,1

0.48447,0.56991,1

0.25476,0.34596,1

0.21726,0.28641,1

0.67078,0.46538,1

0.3815,0.4622,1

0.53838,0.32774,1

0.4849,0.26071,1

0.37095,0.38809,1

0.54527,0.63911,1

0.32149,0.12007,1

0.42216,0.61666,1

0.10194,0.060408,1

0.15254,0.2168,1

0.45558,0.43769,1

0.28488,0.52142,1

0.27633,0.21264,1

0.39748,0.31902,1

0.5533,1,0

0.44274,0.59205,0

0.85176,0.6612,0

0.60436,0.86605,0

0.68243,0.48301,0

1,0.76815,0

0.72989,0.8107,0

0.67377,0.77975,0

0.78761,0.58177,0

0.71442,0.7668,0

0.49379,0.54226,0

0.78974,0.74233,0

0.67905,0.60921,0

0.6642,0.72519,0

0.79396,0.56789,0

0.70758,0.76022,0

0.59421,0.61857,0

0.49364,0.56224,0

0.77707,0.35025,0

0.79785,0.76921,0

0.70876,0.96764,0

0.69176,0.60865,0

0.66408,0.92075,0

0.65973,0.66666,0

0.64574,0.56845,0

0.89639,0.7085,0

0.85476,0.63167,0

0.62091,0.80424,0

0.79057,0.56108,0

0.58935,0.71582,0

0.56846,0.7406,0

0.65912,0.71548,0

0.70938,0.74041,0

0.59154,0.62927,0

0.45829,0.4641,0

0.79982,0.74847,0

0.60974,0.54757,0

0.68127,0.86985,0

0.76694,0.64736,0

0.69048,0.83058,0

0.68122,0.96541,0

0.73229,0.64245,0

0.76145,0.60138,0

0.58985,0.86955,0

0.73145,0.74516,0

0.77029,0.7014,0

0.73156,0.71782,0

0.44556,0.57991,0

0.85275,0.85987,0

0.51912,0.62359,0

import numpy as np

def softmax(L):

expL = np.exp(L)

sumExpL = sum(expL)

result = []

for i in expL:

result.append(i\*1.0/sumExpL)

return result

# Note: The function np.divide can also be used here, as follows:

# def softmax(L):

# expL = np.exp(L)

# return np.divide (expL, expL.sum())