```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


parameters = {"axes.labelsize": 20, "axes.titlesize": 30, 'xtick.labelsize': 12, "ytick.labelsize": 12, "legend.fontsize": 12}

plt.rcParams.update(parameters)


d_M = pd.read_csv('C:₩₩Users₩₩kim07₩₩Downloads₩₩NN_data.csv')
# 데이터파일 불러오기

d_M = d_M.to_numpy(dtype=float)


'''받아온 파일의 데이터 x와 y로 자동 분류 해주는 함수'''

def make_input_output(M):


    for i in range(len(M[0, :])):                                   #M의 column 수만큼 반복

        if i == 0:

            x_matrix = M[:, 0]                                      #M의 첫번째 column 성분 값들 x_matrix에 저장

        elif i < (len(M[0, :]) - 1):

            x_matrix = np.column_stack([x_matrix, M[:, i]])         #M의 마지막 column 성분 제외한 값들 x_matrix에 저장

        else:

            y = M[:, i]                                             #M의 마지막 column 성분 y에 저장

    y = y.reshape(len(y), 1)                                        #y
```

size 다듬기

```
    x_matrix_t = np.transpose(x_matrix)                                    #한
데이터에 대한 특징들 한 column에 나타나기 위해 transpose

    y_t = np.transpose(y)                                                   #위
와 동일

    return x_matrix_t, y_t


'''데이터의 class수 세는 함수'''

def y_class(y):


    y_class = np.unique(y)
#class 수 계산

    Q = len(y_class)
#numpy array로 받아지기 때문에 길이를 셈


    return Q


'''데이터의 특징 수 세는 함수'''

def ch_count(y):


    Q = len(y)                                                              #
받아온 데이터의 열 개수를 셈


    return Q


'''One-Hot Encoding 구현 함수'''

def One_Hot_Encoding(y):
```

```
    Q = y_class(y)
#class 수를 셈
    print("class 수:", Q)

    y_vector = np.zeros([Q, len(y[0, :])])                                    #Q × 샘
플 수의 zero 벡터 생성

    # i번 index의 y데이터가 1~6 각각에 해당될 때 그 자리에 1 저장
    for i in range(len(y[0, :])):
        if y[0, i] == 1:
            y_vector[0, i] = 1
        elif y[0, i] == 2:
            y_vector[1, i] = 1
        elif y[0, i] == 3:
            y_vector[2, i] = 1
        elif y[0, i] == 4:
            y_vector[3, i] = 1
        elif y[0, i] == 5:
            y_vector[4, i] = 1
        else:
            y_vector[5, i] = 1

    return y_vector

'''row기반 dummy추가해주는 함수'''
```

```
def add_dummy(x):

    x_dummy = np.ones(len(x[0, :]))                                    #입
력데이터 x의 길이만큼 dummy 생성
    x = np.row_stack([x, x_dummy])
#row방향으로 쌓음

    return x

'''sigmoid 구현 함수'''
def sigmoid_function(z):

    return(1/(1 + np.exp(-z)))

'''0.5보다 클 때 1로 반환해주는 함수'''
def classification_data(p):

    return (p >= 0.5).astype(int)
'''0 ~ 1 사이를 class수만 큼 나누어 그에 해당하는 값 저장함수(Rule - Based)'''
def classification_data_division(y, Q):

    p = np.zeros(len(y[0, :]))
    for i in range(len(y[0, :])):
        if y[0, i] <= 1 / Q:
            p[i] = 1
        elif (y[0, i] > 1 / Q) & (y[0, i] <= 2 / Q):
```

```
            p[i] = 2
        elif (y[0, i] > 2 / Q) & (y[0, i] <= 3 / Q):
            p[i] = 3
        elif (y[0, i] > 3 / Q) & (y[0, i] <= 4 / Q):
            p[i] = 4
        elif (y[0, i] > 4 / Q) & (y[0, i] <= 5 / Q):
            p[i] = 5
        else:
            p[i] = 6
    return p

'''대표값 찾아서 1로 만들어주는 함수'''
def classification_data_max(y):

    p = np.zeros([len(y[:, 0]), len(y[0, :])])                                    #받아온 y데
이터의 row와 column 크기만큼 요소가 0인 matrix 생성

    #y의 j열, i행 해당하는 값이 i행의 최댓값과 같을 때 p[j, i]에 1 저장
    for i in range(len(y[0, :])):
        for j in range(len(y[:, 0])):
            if y[j, i] == np.max(y[:, i]):
                p[j,i] = 1

    return p
```

```
'''데이터 정확도 함수'''

def data_accuracy(y_h, y):                                                    #한
데이터에 대한 성분 row로 나열한 데이터기준


    count = 0
#count 기능 이용할 변수 0으로 초기화


    for i in range(len(y_h[0, :])):                                           #예측 데
이터 column 성분만큼 반복

        if (y_h[:, i] == y[:, i]).all():                                      #받아온 데
이터와 예측 데이터의 같은 column의 row성분 값이 모두 같은지 확인

            count += 1                                                        #
위 조건에 해당할 때 count


    accuracy = count / len(y_h[0, :])
#count 된 수를 예측데이터 column 개수만큼 나눠줌


    return accuracy



#
============================================================
==================

# def Perceptron(row, column, input_data):

#     input_data = add_dummy(input_data)

#     w = np.random.rand(row, column) - 0.5

#     output = np.dot(w, input_data)
```

```
#       act_output = sigmoid_function(output)
#       return act_output
#
# def Two_Layer_Neural_Network(x_input, y_data):
#       M = ch_class(x_input)
#       Q = ch_class(y_data)
#       L = ch_class(x_input) + 1                   #hidden layer node 수(사용자 지정)
#
#       b_matrix = Perceptron(L, M + 1, x_input)
#
#       y_hat = Perceptron(Q, L + 1, b_matrix)
#       return y_hat
#
# perceptron 따로 계산하는 코드 => 내용 이해했다고 보여주기 위해 진행 과정이 잘 보이는 아래의 함수로 씀
#
==========================================================================================

'''Two Layer Neural Network 구현 함수'''
def Two_Layer_Neural_Network(x_input, y_data, L):
#Hidden Layer의 node 수 지정

    M = ch_count(x_input)
#input 특징 수 세기
    Q = ch_count(y_data)
```

#output 특징 수 세기

    x_input = add_dummy(x_input)
#input data에 dummy 추가

    v_matrix = np.random.rand(L, M + 1) - 0.5 #사
용자 지정 hidden layer의 node, input데이터 특징 수 + 1만큼 weight 생성

    alpha = np.dot(v_matrix, x_input) #v와
xinput을 곱해 alpha를 구함

    b_matrix = sigmoid_function(alpha)
#alpha를 sigmoid function에 넣어 b 구함

    b_matrix = add_dummy(b_matrix)
#b에 dummy 추가

    w_matrix = np.random.rand(Q, L + 1) - 0.5
#output 수, hidden layer node 수 + 1만큼 weight 생성

    beta = np.dot(w_matrix, b_matrix) #w
와 b 곱해서 beta 구함

    y_hat = sigmoid_function(beta)
#beta를 sigmoid function에 넣어 y_hat 구함

    return y_hat
#y_hat 반환

'''main'''

x_matrix, y_vector = make_input_output(d_M) #받
아온 데이터를 input과 output으로 나눔

y_data = One_Hot_Encoding(y_vector)
#output data를 one-hot-encoding 방식으로 변환

y_hat = Two_Layer_Neural_Network(x_matrix, y_data, 9)
#hidden layer의 node 수는 9개이고 forward propagation한 값 y_hat에 저장

p = classification_data(y_hat)
#y_hat 값을 0.5 기준으로 0과 1로 분류

p_m = classification_data_max(y_hat)
#y_hat 값을 대푯값 구해서 그 index만 1로 만들어서 분류

accuracy1 = data_accuracy(p, y_data) #0.5 기준으로 분류한 것 정확도 계산

accuracy2 = data_accuracy(p_m, y_data) #대푯값을 구한 index 1로 만든 것 정확도 계산

#직선 분리 형식

x_matrix = add_dummy(x_matrix)
#x_matrix에 dummy 추가

w = np.random.rand(1,len(x_matrix)) - 0.5
#x_matrix 길이만큼 weight 생성

Q = y_class(y_vector)
#y_vector class 수

z = np.dot(w, x_matrix) #선형조합으로 z 구함

p_r = sigmoid_function(z) #z

sigmoid function에 넣어 사후 확률 구함

y_h = classification_data_division(p_r, Q) #0~1사
이를 class수만큼 나눠서 분류하는 함수에 사후 확률 넣어서 y_h 구함

y_h = np.reshape(y_h, [1, len(y_h)])
#(1800,1) 을 (1, 1800)으로 바꿔줌

accuracy3 = data_accuracy(y_h, y_vector) #y_h
의 정확도 계산

print("0.5 기준 분류 정확도:", accuracy1)

print("Two Layer Neural Network 정확도:", accuracy2)

print("Rule-based 분류 정확도:", accuracy3)