

```

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)

'''training set, validation set, test set 나누는 함수'''

def data_division(n_data, Tr_rate, V_rate, Te_rate):

    np.random.shuffle(n_data)                                #데이터
    섞기

    tr_index = int(len(n_data) * Tr_rate / 10)               #Tr_set 비율
    만큼 데이터 index 양 확인

    v_index = int(len(n_data) * V_rate / 10)                 #V_set 비
    울만큼 데이터 index 양 확인

    te_index = int(len(n_data) * Te_rate / 10)               #Te_set 비
    울만큼 데이터 index 양 확인

    #비율대로 data 나누기

    tr_set = n_data[0:tr_index]

    v_set = n_data[tr_index : tr_index + v_index]

    te_set = n_data[tr_index + v_index : tr_index + v_index + te_index]

    return tr_set, v_set, te_set

'''dummy 추가 함수'''

```

```
def add_dummy(x):
    x_dummy = np.ones(len(x)) #
    입력데이터 x의 길이만큼 dummy 생성
    x = np.column_stack([x, x_dummy])
    return x
```

'''sigmoid함수 '''

```
def sigmoid_funtion(z):
    return(1/(1 + np.exp(-z)))
#sigmoid 함수 꼴 바로 return값으로 넣어줌
```

```
def data_accuracy(y_h, y):
    accuracy = np.mean(y_h == y)
    return accuracy
```

'''logistic regression 구현 함수'''

```
def logistic_regression(epoch, LR, x, y): #변화 시
    킬 variable
```

```
    w_hist = [] #w
    값 저장할 list

    CEE_hist = []
    #MSE값 저장할 list

    ACC_hist = []
    #ACC값 저장할 list
```

```

#epoch만큼 반복해 weight 업데이트

for i in range(epoch):

    if i == 0:

        w = np.random.rand(len(x[0,:])) * 5                                #w0,
w1값을 0과 5사이에 생성

        w = np.reshape(w, [-1, 1])

        z = np.dot(x, w)

        #y_hat 행렬곱 => 결과값이 vector

        p = sigmoid_funtion(z)
#sigmoid 함수에 z 넣어서 사후 확률 얻기

        y_h = classification_data(p)                                        #0과
1로 분류

        y_h.reshape(-1, 1)

        CEE = -np.mean(y * np.log(p) + (1 - y) * np.log(1 - p))
#CEE구하기

        accuracy = data_accuracy(y_h, y)
#y_h와 y가 같을 때의 평균을 구함

        w_hist.append(w)
#w_hist에 w값 저장

        CEE_hist.append(CEE)
#MSE_hist에 MSE값 저장

        ACC_hist.append(accuracy)
#ACC_hist에 ACC값 저장

```

```

        x_t = np.transpose(x)                                #x
transpose
        dif = np.dot(x_t, (p - y)) / len(y)                 #w의 기
올기 구하기 => size가 2인 vector(w0, w1)

        w = w - IR * dif
#weight update

    return w, w_hist, CEE_hist, ACC_hist

def logistic_regression_with_MSE(epoch, IR, x, y):
#변화 시킬 variable

    w_hist = []                                              #w
값 저장할 list

    MSE_hist = []
#MSE값 저장할 list

    ACC_hist = []
#ACC값 저장할 list

    #epoch만큼 반복해 weight 업데이트

    for i in range(epoch):

        if i == 0:

            w = np.random.rand(len(x[0,:])) * 5             #w0,
w1값을 0과 5사이에 생성

```

```

w = np.reshape(w, [-1, 1])

z = np.dot(x, w)
#y_hat 행렬곱 => 결과값이 vector

p = sigmoid_funtion(z)
#sigmoid 함수에 z 넣어서 사후 확률 얻기

y_h = classification_data(p)
1로 분류
#0과

y_h.reshape(-1, 1)

error = p - y

MSE = np.mean((error)**2)

accuracy = data_accuracy(y_h, y)
#y_h와 y가 같을 때의 평균을 구함

w_hist.append(w)
#w_hist에 w값 저장

MSE_hist.append(MSE)
#MSE_hist에 MSE값 저장

ACC_hist.append(accuracy)
#ACC_hist에 ACC값 저장

x_t = np.transpose(x)

w_dif = 2* np.dot(x_t, error)/len(y)
size가 2인 vector(w0, w1)
#w의 기울기 구하기 =>

```

```
w = w - IR * w_dif
```

```
#weight update
```

```
return w, w_hist, MSE_hist, ACC_hist
```

```
'''데이터 0, 1 분류 함수(내가 구현한거)'''
```

```
# def classification_data(p):
```

```
#     y_hat = []
```

```
#     for i in range(len(p)):
```

```
#         if (p[i] < 0.5):
```

```
#             y_hat = np.append(y_hat, 0)
```

```
#         else:
```

```
#             y_hat = np.append(y_hat, 1)
```

```
#     return y_hat
```

```
'''gpt가 알려준 성능 더 좋은 분류 방법'''
```

```
def classification_data(p):
```

```
    return (p >= 0.5).astype(int)
```

```
#p의 값에 따라 true, false 생성,
```

```
astype(int)으로 true면 1, false면 0 생성
```

```
M = pd.read_csv('C:\Users\wkim07\Downloads\logistic_regression_data.csv') #
```

```
데이터파일 불러오기
```

```
M = M.to_numpy(dtype=float)
```

```
#tr, v, te set 비율 설정
```

```
Tr_rate = 7
```

```
V_rate = 0
```

```
Te_rate = 3
```

```
tr_m, v_m, te_m = data_division(M, Tr_rate, V_rate, Te_rate)
```

```
#data 분
```

```
할하는 함수 사용
```

```
tr_x_matrix = tr_m[:, 1:3]
```

```
#training 데이터 1~2행의 데이터 저장
```

```
tr_x_matrix = add_dummy(tr_x_matrix)
```

```
#dummy 추가
```

```
tr_y = tr_m[:, 3]
```

```
#training 데이터 3행 데이터 저장
```

```
tr_y = np.reshape(tr_y, [len(tr_y), 1])
```

```
#크기 맞
```

```
춰주기
```

```
te_x_matrix = te_m[:, 1:3]
```

```
#test
```

```
데이터 1~2행의 데이터 저장
```

```
te_x_matrix = add_dummy(te_x_matrix)
```

```
#dummy 추가
```

```
te_y = te_m[:, 3]
```

```
#test 데이터 3행 데이터 저장
```

```
te_y = np.reshape(te_y, [len(te_y), 1])
```

#epoch, learning rate 설정

epoch = 3000

IR = 0.05

#1.9부터 불안정해짐

#logistic regression 함수 사용

tr\_w, tr\_w\_hist, tr\_CEE\_hist, tr\_ACC\_hist = logistic\_regression(epoch, IR, tr\_x\_matrix, tr\_y)

tr\_w\_hist = np.array(tr\_w\_hist) #그래

프에 그리기 위해 numpy 배열로 바꿔줌

te\_z = np.dot(te\_x\_matrix, tr\_w) #학습

한 weights로 test set에 대한 z값 구하기

te\_p = sigmoid\_funtion(te\_z) #사

후확률 p 구하기

y\_hat = classification\_data(te\_p) #0.5를

기준으로 1과 0으로 분류

te\_accuracy = data\_accuracy(y\_hat, te\_y)

print("CEE로 학습한 머신의 test set Accuracy: ", te\_accuracy)

#분류 그래프 그리기 위한 test set에 대한 x 값구하기

tr\_x\_min = np.min(tr\_x\_matrix) - 1 #최소

test x값에서 1더 작은 값을 저장

tr\_x\_max = np.max(tr\_x\_matrix) + 1 #최대

test x값에서 4더 큰 값을 저장(분류 직선 길이 때문)

tr\_x\_step = 0.1



```
tr_x_graph = np.arange(tr_x_min, tr_x_max, tr_x_step) #그래프  
용 x
```

```
tr_x1 = -(tr_w[0] / tr_w[1]) * tr_x_graph - (tr_w[2] / tr_w[1]) #분류 그  
래프
```

#분류 그래프 그리기 위한 test set에 대한 x 값구하기

```
te_x_min = np.min(te_x_matrix) - 1 #최소  
test x값에서 1더 작은 값을 저장
```

```
te_x_max = np.max(te_x_matrix) + 1 #최대  
test x값에서 4더 큰 값을 저장(분류 직선 길이 때문)
```

```
te_x_step = 0.1
```

```
te_x_graph = np.arange(te_x_min, te_x_max, te_x_step) #그래  
프용 x
```

```
te_x1 = -(tr_w[0] / tr_w[1]) * te_x_graph - (tr_w[2] / tr_w[1]) #분류 그  
래프
```

#epoch에 따른 weight 변화 그래프

```
plt.figure()
```

```
plt.plot(tr_w_hist[:, 0], 'r-o', markevery = 100)
```

```
plt.plot(tr_w_hist[:, 1], 'b-^', markevery = 100)
```

```
plt.plot(tr_w_hist[:, 2], 'g-x', markevery = 100)
```

```
plt.legend(["w0", "w1", "w2"])
```

```
plt.xlabel("epoch")
```

```
plt.ylabel("weights")
```

```
plt.grid()
```

```
#epoch에 따른 Cross Entropy Error와 Accuracy의 변화 그래프
```

```
f, axes1 = plt.subplots(2, 1)
```

```
axes1[0].plot(tr_CEE_hist, '-x', markevery = 100)
```

```
# axes[0].set_title("Cross Entropy Error")
```

```
axes1[0].set_xlabel("epoch")
```

```
axes1[0].set_ylabel("CEE")
```

```
axes1[0].grid()
```

```
axes1[1].plot(tr_ACC_hist, '-x', markevery = 100)
```

```
axes1[1].grid()
```

```
axes1[1].set_xlabel("epoch")
```

```
axes1[1].set_ylabel("accuracy")
```

```
#test set의 분류 정확도 막대 그래프
```

```
plt.figure()
```

```
plt.bar(["Test Accuracy"], [te_accuracy])
```

```
plt.ylim(0, 1)
```

```
plt.title("Test Set Accuracy")
```

```
plt.ylabel("Accuracy")
```

```
plt.grid()
```

```
plt.show()
```

#training set에 대한 Decision Boundary 그래프

```
plt.figure()
```

```
for i in range(len(tr_y)):
```

```
    if tr_y[i] == 0:
```

```
        sc0 = plt.scatter(tr_x_matrix[i, 0], tr_x_matrix[i, 1], c = 'red', marker = 'x')
```

```
    else:
```

```
        sc1 = plt.scatter(tr_x_matrix[i, 0], tr_x_matrix[i, 1], c = 'blue', marker = 'o')
```

```
line = plt.plot(tr_x_graph, tr_x1)[0]
```

```
plt.title("training set")
```

```
plt.xlabel("x0")
```

```
plt.ylabel("x1")
```

```
plt.grid()
```

```
plt.legend(handles = [sc0, sc1, line], labels = ["0", "1", "decision boundary"], loc = "lower  
right")
```

#test set에 대한 Decision Boundary 그래프

```
plt.figure()
```

```
for i in range(len(te_y)):
```

```
    if te_y[i] == 0:
```

```
        sc0 = plt.scatter(te_x_matrix[i, 0], te_x_matrix[i, 1], c = 'black', marker = 'x')
```

```
    else:
```

```
        sc1 = plt.scatter(te_x_matrix[i, 0], te_x_matrix[i, 1], c = 'green', marker = 'o')
```

```
line = plt.plot(te_x_graph, te_x1)[0]
```

```
plt.title("test set")
```

```
plt.xlabel("x0")
```

```
plt.ylabel("x1")
```

```
plt.grid()
```

```
plt.legend(handles = [sc0, sc1, line], labels = ["0", "1", "decision boundary"], loc = "lower  
right")
```

```
'''MSE이용한 것으로 다시 학습'''
```

```
M_epoch = 6000
```

```
M_IR = 0.9      #0.4부터 불안정하기 시작 0.8 까지
```

```
#logistic regression 함수 사용
```

```
M_tr_w, M_tr_w_hist, tr_MSE_hist, M_tr_ACC_hist = logistic_regression_with_MSE(M_epoch,  
M_IR, tr_x_matrix, tr_y)
```

```
M_tr_w_hist = np.array(M_tr_w_hist)
```

```
#그래프에 그리기 위해 numpy 배열로 바꿔줌
```

```
M_te_z = np.dot(te_x_matrix, M_tr_w)
```

```
#학습한 weights로 test set에 대한 z값 구하기
```

```
M_te_p = sigmoid_funtion(M_te_z)
```

```
#사후확률 p 구하기
```

```
M_y_hat = classification_data(M_te_p)
```

```
#0.5를 기준으로 1과 0으로 분류
```

```
M_te_accuracy = data_accuracy(M_y_hat, te_y)
```

```
print("MSE로 학습한 머신의 test set Accuracy: ", M_te_accuracy)
```

```
#분류 그래프 그리기 위한 test set에 대한 x 값구하기
```

```
M_tr_x_min = np.min(tr_x_matrix) - 1
```

```
#최소
```

test x값에서 1더 작은 값을 저장

M\_tr\_x\_max = np.max(tr\_x\_matrix) + 1

#최

대 test x값에서 4더 큰 값을 저장(분류 직선 길이 때문)

M\_tr\_x\_step = 0.1

M\_tr\_x\_graph = np.arange(M\_tr\_x\_min, M\_tr\_x\_max, M\_tr\_x\_step)

#그래프용 x

M\_tr\_x1 = -(M\_tr\_w[0] / M\_tr\_w[1]) \* M\_tr\_x\_graph - (M\_tr\_w[2] / M\_tr\_w[1])

#분류 그래프

#분류 그래프 그리기 위한 test set에 대한 x 값구하기

M\_te\_x\_min = np.min(te\_x\_matrix) - 1

#최

소 test x값에서 1더 작은 값을 저장

M\_te\_x\_max = np.max(te\_x\_matrix) + 1

#최

대 test x값에서 4더 큰 값을 저장(분류 직선 길이 때문)

M\_te\_x\_step = 0.1

M\_te\_x\_graph = np.arange(M\_te\_x\_min, M\_te\_x\_max, M\_te\_x\_step)

#그래프용 x

M\_te\_x1 = -(M\_tr\_w[0] / M\_tr\_w[1]) \* M\_te\_x\_graph - (M\_tr\_w[2] / M\_tr\_w[1])

plt.figure()

plt.plot(M\_tr\_w\_hist[:, 0], 'r-o', markevery = 100)

plt.plot(M\_tr\_w\_hist[:, 1], 'b-^', markevery = 100)

plt.plot(M\_tr\_w\_hist[:, 2], 'g-x', markevery = 100)

plt.legend(["w0", "w1", "w2"])

plt.xlabel("epoch")

plt.ylabel("weights")

```
plt.grid()
```

```
f, axes2 = plt.subplots(2, 1)
```

```
axes2[0].plot(tr_MSE_hist, '-x', markevery = 100)
```

```
axes2[0].set_xlabel("epoch")
```

```
axes2[0].set_ylabel("MSE")
```

```
axes2[0].grid()
```

```
axes2[1].plot(M_tr_ACC_hist, '-x', markevery = 100)
```

```
axes2[1].grid()
```

```
axes2[1].set_xlabel("epoch")
```

```
axes2[1].set_ylabel("accuracy")
```

```
#test set의 분류 정확도 막대 그래프
```

```
plt.figure()
```

```
plt.bar(["Test Accuracy"], [M_te_accuracy])
```

```
plt.ylim(0, 1)
```

```
plt.title("Test Set Accuracy")
```

```
plt.ylabel("Accuracy")
```

```
plt.grid()
```

```
plt.show()
```

```
#training set에 대한 Decision Boundary 그래프
```

```
plt.figure()
```

```

for i in range(len(tr_y)):
    if tr_y[i] == 0:
        sc0 = plt.scatter(tr_x_matrix[i, 0], tr_x_matrix[i, 1], c = 'red', marker = 'x')
    else:
        sc1 = plt.scatter(tr_x_matrix[i, 0], tr_x_matrix[i, 1], c = 'blue', marker = 'o')
line = plt.plot(M_tr_x_graph, M_tr_x1)[0]
plt.title("training set")
plt.xlabel("x0")
plt.ylabel("x1")
plt.grid()
plt.legend(handles = [sc0, sc1, line], labels = ["0", "1", "decision boundary"], loc = "lower
right")

```

#test set에 대한 Decision Boundary 그래프

```

plt.figure()
for i in range(len(te_y)):
    if te_y[i] == 0:
        sc0 = plt.scatter(te_x_matrix[i, 0], te_x_matrix[i, 1], c = 'black', marker = 'x', label
= "class 0")
    else:
        sc1 = plt.scatter(te_x_matrix[i, 0], te_x_matrix[i, 1], c = 'green', marker = 'o',
label = "class 1")
line = plt.plot(M_te_x_graph, M_te_x1, label = "decision boundary")[0]
plt.title("test set")
plt.xlabel("x0")
plt.ylabel("x1")

```

```
plt.grid()
```

```
plt.legend(handles = [sc0, sc1, line], labels = ["0", "1", "decision boundary"], loc = "lower  
right")
```