



전자공학과 2020142001 곽종근



중간 프로젝트 심근경색 데이터 Set 예측

제출일: 2024.06.08.

데이터 전처리 & 특징추출에 대한 코드

```
# 데이터 전처리 & 특징추출
def select_features(file_path):
    #데이터 불러오기
    path = directory + "heart_disease_new.csv"
    df = pd.read_csv(path) # pandas를 사용하여 csv 파일 읽기
    dataset = np.array(df) # pandas DataFrame을 numpy 배열로 변환

    # 데이터 섞기
    np.random.shuffle(dataset)

    #===== 전처리=====

    #path데이터(heart_disease_new.csv)에서 ,를 사용해 구분하고 문자열로 1행 분리
    # csv 파일의 첫 번째 행을 열 이름으로 사용하여 데이터 읽기
    column_names = np.genfromtxt(path, delimiter=',', dtype=str, max_rows=1)

    # 열 이름을 기준으로 인덱스를 찾아서 저장하며
    # 각 특징들의 인덱스 위치가 저장됨
    gender_idx = np.where(column_names == "gender")[0][0]
    neuro_idx = np.where(column_names == "a neurological disorder")[0][0]
    target_idx = np.where(column_names == "heart disease")[0][0]
    age_idx = np.where(column_names == "Age")[0][0]
    hbp_idx = np.where(column_names == "High blood pressure")[0][0]
    chol_idx = np.where(column_names == "Cholesterol")[0][0]
    height_idx = np.where(column_names == "height")[0][0]
    weight_idx = np.where(column_names == "Weight")[0][0]
    bmi_idx = np.where(column_names == "BMI")[0][0]
    bpm_idx = np.where(column_names == "BPMeds")[0][0]
    bloodsugar_idx = np.where(column_names == "blood sugar levels")[0][0]
    meat_idx = np.where(column_names == "meat intake")[0][0]
    smoke_idx = np.where(column_names == "Smoking")[0][0]

    # gender특징에서 female = 0, male = 1
    dataset[:, gender_idx] = np.where(dataset[:, gender_idx] == 'female', 0, 1)

    # a neurological disorder특징에서 yes = 1, no = 0
    dataset[:, neuro_idx] = np.where(dataset[:, neuro_idx] == 'yes', 1, 0)

    # heart disease결과에서 yes = 1, no = 0
    dataset[:, target_idx] = np.where(dataset[:, target_idx] == 'yes', 1, 0)

    # nan값인 결측값을 처리하는데 이를 각 열에 대한 평균값으로 대체함
    for i in range(dataset.shape[1]):
        col = dataset[:, i].astype(float)
        mean_val = np.nanmean(col) # 결측치를 제외한 평균값 계산
        col[np.isnan(col)] = mean_val # 결측치를 평균값으로 대체
        dataset[:, i] = col

    # 데이터 형식을 float로 변환
    dataset = dataset.astype(float)
```

데이터 전처리 부분에서는
csv파일을 불러와 numpy로 바꿔주고,

각 특징을 슬라이싱 하는 방법으로 np.genfromtxt()를 이용하여
Column에 쓰여있는 특징이름의 인덱스를 list로 만들어주며,
각 특징의 index를 특징_idx에 저장

Gender -> female = 0, male = 1

a neurological disorder -> yes = 1, no = 0

heart disease -> yes = 1, no = 0 으로 데이터 수정

For문을 이용하여 nan값으로 적힌 값들을 각 열의 평균값으로 변경

데이터 형식을 float로 변경

데이터 전처리 & 특징추출에 대한 코드

```
# 편향 조정 실제 코드에서는 필요가 없을 수 있으나 heart_disease_new파일에서
# 결과 yes, no의 비율이 500:2500으로 no인 0으로만 분류하더라도(분류가 안되어도)
# 정확도가 높게 나오게 되므로 확실한 정확도 계산을 위해 5:5로 비율을 맞춰줌
yes_data = dataset[dataset[:, target_idx] == 1]
no_data = dataset[dataset[:, target_idx] == 0]
np.random.shuffle(no_data)
no_data = no_data[:500] # no 데이터를 500개로 샘플링
balanced_data = np.vstack((yes_data, no_data))
#=====#
# y_data 추출
# y_data = dataset[:, -1] # 타겟 값 추출

np.random.shuffle(balanced_data) # 데이터를 섞기

# y_data 추출
y_data = balanced_data[:, target_idx] # 타겟 값 추출

# 특징 추출 (상관관계에 기반하여 조합)
# 다양한 특징을 시도하였고 최종적으로 1, 5, 12를 사용
# 키가 상관관계가 매우 커서 사용
feature_1 = balanced_data[:, height_idx]
# 특징 5번 이 데이터에서 구해져있는 bmi는 값이 미소한 것들이 많아 따로 bmi를 구해보았다.
feature_5 = balanced_data[:, weight_idx] / ((balanced_data[:, height_idx]/100)**2)
# 특징 12번 데이터에서 흡연이 yes이면 무조건 고기를 섭취하므로 두 값을 더하고, 큰 상관관계인 키/5를 빼주어 특징을 구현
feature_12 = ((balanced_data[:,smoke_idx]*10) + (balanced_data[:,meat_idx] * 10)) - balanced_data[:,height_idx] / 5

# 선택한 특징들을 결합
selected_features = np.column_stack((feature_1, feature_5, feature_12))
# 특징 1개
# selected_features = feature_x

# 특징 데이터 마지막 열에 y값 추가
features = np.column_stack((selected_features, y_data))

return features
```

실제 코드에서는 필요가 없을 수 있으나 heart_disease_new파일에서 yes, no의 비율이 500:2500으로 no로만 분리하여도 정확도가 높게 나와 정확한 정확도 계산을 위해 5:5 비율로 이를 수정
특징은 3개 사용 -> 키, 따로 구한 bmi, 흡연과 고기섭취의 상관관계와 키의 차이들을 결합하고 y데이터를 추가

데이터 전처리 & 특징추출에 대한 코드

```
#테스트 데이터와 트레인 데이터를 7:3으로 분리하여 각 정확도를 판별
train_data, test_data = aug_data(features, 0.7, 0.3)
```

```
# 데이터 분리
```

```
x_train = train_data[:, :-1]
y_train = train_data[:, -1].reshape(-1, 1)
```

```
x_test = test_data[:, :-1]
y_test = test_data[:, -1].reshape(-1, 1)
```

```
# 입력 속성 수와 출력 클래스 수 추출
# 아웃풋은 0,1이므로 output_size = 1
```

```
M = x_train.shape[1]
output_size = 1
```

```
#히든 사이즈 = 2
hidden_size = 2
```

```
# v,m의 가중치를 정규분포로 random 선언
```

```
# v = L by M+1의 크기로 w_hidden
```

```
v = np.random.randn(hidden_size, M + 1)*0.01
```

```
# w = 1 by L+1의 크기로 w_output
```

```
w = np.random.randn(output_size, hidden_size + 1) *0.01
```

```
#최적의 학습률을 돌려보았더니 0.0006이 최적으로 나왔고, 매우 민감하
```

```
learning_rate = 0.0006
```

```
epochs = 500
```

```
x_train_with_dummy = np.hstack((x_train, np.ones((len(x_train), 1))))
```

```
x_test_with_dummy = np.hstack((x_test, np.ones((len(x_test), 1))))
```

```
total_samples = len(x_train)
```

```
accuracy_list = []
```

```
mse_list = []
```

```
mse_test_list = []
```

```
test_accuracy_list = []
```

```
best_accuracy = 0
```

```
best_v = np.copy(v)
```

```
best_w = np.copy(w)
```

```
# 학습시키는 코드, batch_size를 1로 설정하여 전체를 돌리면 1에폭이 되도록 한다.
```

```
for epoch in range(epochs):
```

```
    for step in range(total_samples):
```

```
        A, b, b_with_dummy, B, y_hat = forward_propagation(x_train_with_dummy[step:step+1], v, w)
```

```
        wmse, vmse = backward_propagation(x_train_with_dummy[step:step+1], y_train[step:step+1], A, b, b_with_dummy, B, y_hat, v, w)
```

```
        w -= learning_rate * wmse
```

```
        v -= learning_rate * vmse
```

```
    # 테스트 데이터에 대해 정확도 계산
```

```
    A_test, b_test, b_with_dummy_test, B_test, y_hat_test = forward_propagation(x_test_with_dummy, v, w)
```

```
    y_hat_test_index = (y_hat_test >= 0.5).astype(int).flatten()
```

```
    test_accuracy = np.mean(y_hat_test_index == y_test.flatten())
```

```
    test_accuracy_list.append(test_accuracy)
```

```
    if test_accuracy > best_accuracy:
```

```
        best_accuracy = test_accuracy
```

```
        best_v = np.copy(v)
```

```
        best_w = np.copy(w)
```

```
    A_train, b_train, b_with_dummy_train, B_train, y_hat_train = forward_propagation(x_train_with_dummy, v, w)
```

```
    predicted_labels = (y_hat_train >= 0.5).astype(int).flatten()
```

```
    accuracy = np.mean(predicted_labels == y_train.flatten())
```

```
    accuracy_list.append(accuracy)
```

```
    print(test_accuracy)
```

```
    mse = np.mean((y_hat_train - y_train.T) ** 2)
```

```
    mse_list.append(mse)
```

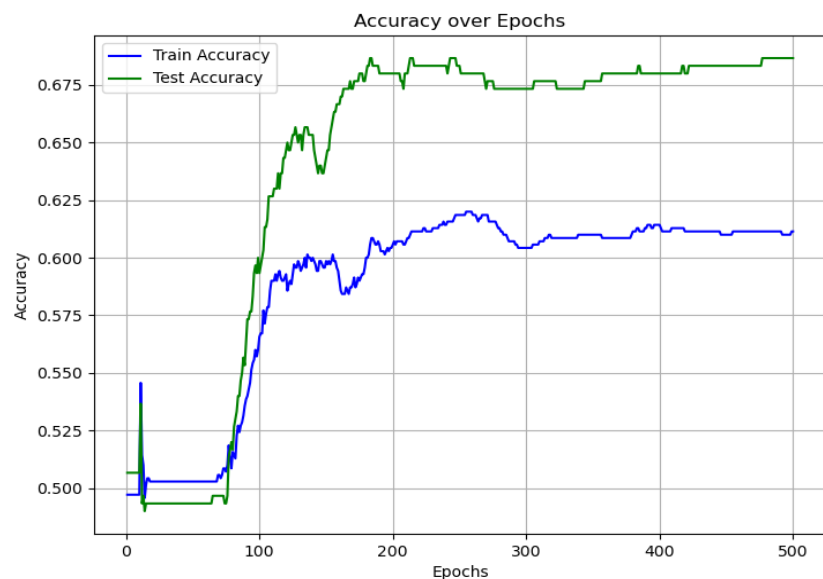
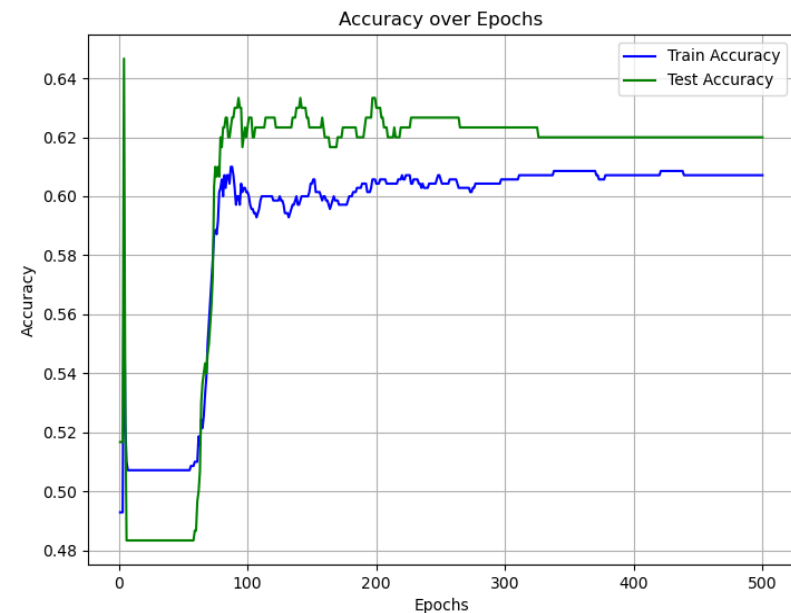
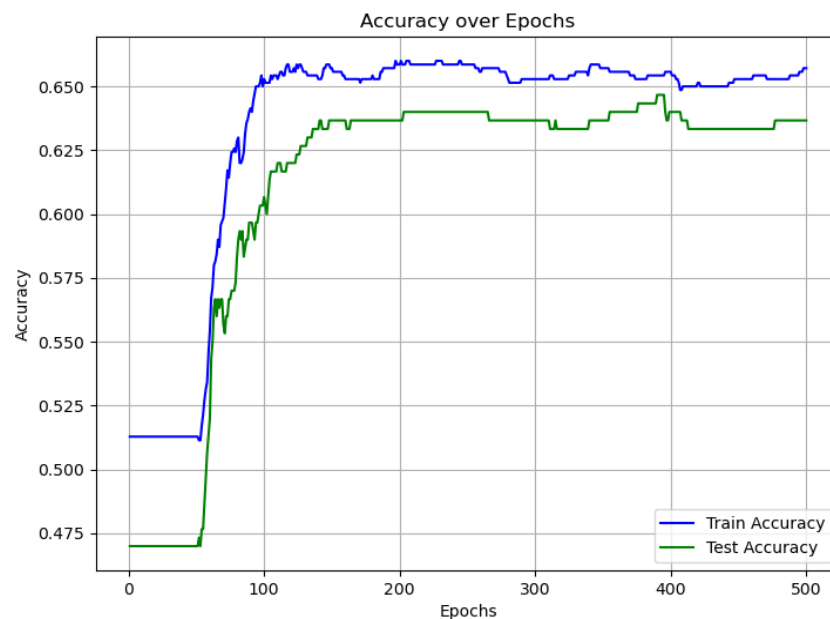
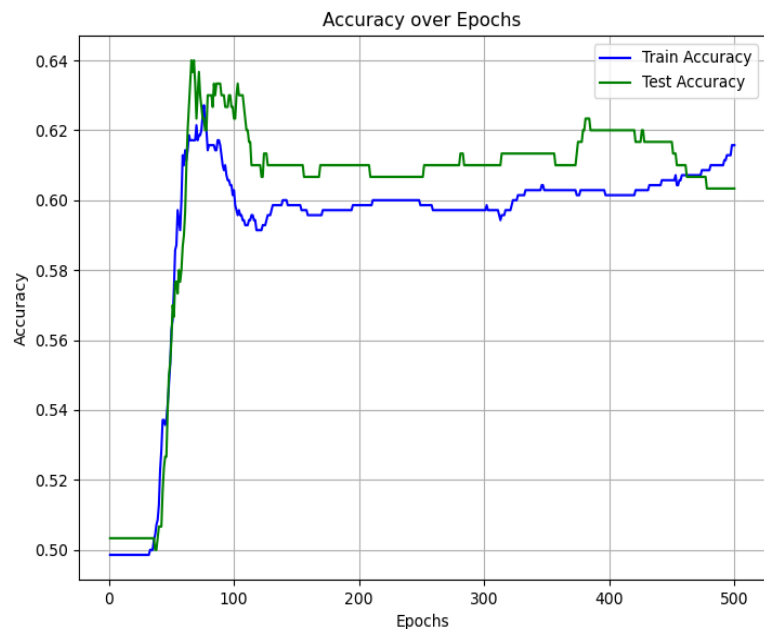
```
# 최적의 가중치로 모델 업데이트
```

```
v = best_v
```

```
w = best_w
```

최종 학습에 있어서 각 노드 수와 learning_rate, epochs는
output_size = 1, hidden_size = 2, learning_rate = 0.0006, epochs = 500
Batch_size는 1로 하여 전체 데이터를 학습하면 1epoch로 진행
0.5 이상이면 1, 미만이면 0으로 분류

특징추출 후 코드 실행, 정확도 판별



같은 코드를 실행함에 있어서 데이터를 랜덤으로 가져오므로 정확도가 계속해서 변화하며 그 가중치들의 값이 다르게 나오게 되고, 여러 번 실행시킨 코드 중 가장 정확도가 높게 나온 w, v 를 사용

가중치 test코드 설명

```
def select_features(file_path):
    df = pd.read_csv(file_path) # pandas를 사용하여 csv 파일 읽기
    dataset = np.array(df) # pandas DataFrame을 numpy 배열로 변환

    # 데이터 섞기
    np.random.shuffle(dataset)

    #===== 전처리=====

    # 파일 경로 데이터에서 ,를 사용해 구분하고 문자열로 1행 분리
    column_names = np.genfromtxt(file_path, delimiter=',', dtype=str, max_rows=1)

    # 열 이름을 기준으로 인덱스 찾아서 저장
    gender_idx = np.where(column_names == "gender")[0][0]
    neuro_idx = np.where(column_names == "a neurological disorder")[0][0]
    target_idx = np.where(column_names == "heart disease")[0][0]
    age_idx = np.where(column_names == "Age")[0][0]
    hbp_idx = np.where(column_names == "High blood pressure")[0][0]
    chol_idx = np.where(column_names == "Cholesterol")[0][0]
    height_idx = np.where(column_names == "height")[0][0]
    weight_idx = np.where(column_names == "weight")[0][0]
    bmi_idx = np.where(column_names == "BMI")[0][0]
    bpm_idx = np.where(column_names == "BPMeds")[0][0]
    bloodsugar_idx = np.where(column_names == "blood sugar levels")[0][0]
    meat_idx = np.where(column_names == "meat intake")[0][0]
    smoke_idx = np.where(column_names == "Smoking")[0][0]

    # gender: female -> 0, male -> 1
    dataset[:, gender_idx] = np.where(dataset[:, gender_idx] == 'female', 0, 1)

    # a neurological disorder: no -> 0, yes -> 1
    dataset[:, neuro_idx] = np.where(dataset[:, neuro_idx] == 'yes', 1, 0)

    # heart disease: yes -> 1, no -> 0
    dataset[:, target_idx] = np.where(dataset[:, target_idx] == 'yes', 1, 0)
```

```
# 결측치 처리 (각 열의 평균값으로 대체)
for i in range(dataset.shape[1]):
    col = dataset[:, i].astype(float)
    mean_val = np.nanmean(col) # 결측치를 제외한 평균값 계산
    col = np.where(np.isnan(col), mean_val, col) # 결측치를 평균값으로 대체
    dataset[:, i] = col

# 데이터 형식을 float로 변환
dataset = dataset.astype(float)

# y_data 추출
y_data = dataset[:, target_idx] # 타겟 값 추출

# 특징 추출 (상관관계에 기반하여 조합)
feature_1 = dataset[:, height_idx]
feature_5 = dataset[:, weight_idx] / ((dataset[:, height_idx]/100)**2) #내가 구한 bmi
feature_12 = ((dataset[:,smoke_idx]*10) + (dataset[:,meat_idx] * 10)) - dataset[:,height_idx] / 5

# 선택한 특징들을 결합
selected_features = np.column_stack((feature_1, feature_5, feature_12))

# 특징 데이터 마지막 열에 y값 추가
features = np.column_stack((selected_features, y_data))

return features

np.random.shuffle(data)

yes_index = np.where(data[:, -1] == 1)[0]
yes_ = data[yes_index, :] # yes 데이터 추출

no_data_index = np.where(data[:, -1] == 0)[0]
no_data = data[no_data_index, :] # no 데이터 추출

yes = 200 # yes와 no의 개수를 결정해서 뽑아온다
no = 200
data_set = np.vstack([yes[:,yes, :], no_data[:,no, :]]) # 뽑아온 yes와 no를 합쳐준다.
```

이전에 사용했던 특징을 추출해 가중치를 구하는 코드에서 원래는 데이터를 균일하게 추출하기 위해 편향조절을 해 주었지만 test코드에서는 이 코드가 필요 없으므로 지워주고 데이터를 섞어 테스트 데이터를 yes:no를 균일하게 뽑아온다.

가중치 test코드

```
def compute_confusion_matrix_with_probabilities(y_true, y_pred, num_classes):
    #각 정확도 값을 추가하기 위해 행렬 크기 변경
    confusion_matrix = np.zeros((num_classes + 1, num_classes + 1))
    #기존 confusion_matrix로 예측과 실제값으로
    for i in range(len(y_true)):
        row_index = int(y_pred[i])
        col_index = int(y_true[i])
        confusion_matrix[row_index, col_index] += 1
    #정확도 class를 만들어 이 값을 행렬에 대입
    class_accuracies = []
    for i in range(num_classes):
        TP = confusion_matrix[i, i]
        total = np.sum(confusion_matrix[i, :num_classes]) + np.sum(confusion_matrix[:num_classes, i]) - TP
        accuracy = TP / total if total != 0 else 0
        class_accuracies.append(accuracy)
        confusion_matrix[i, num_classes] = accuracy # Add row accuracy
        confusion_matrix[num_classes, i] = accuracy # Add column accuracy

    total_correct = np.trace(confusion_matrix[:num_classes, :num_classes])
    total_predictions = np.sum(confusion_matrix[:num_classes, :num_classes])
    overall_accuracy = total_correct / total_predictions if total_predictions != 0 else 0
    confusion_matrix[num_classes, num_classes] = overall_accuracy # Add overall accuracy
    #대각선 전체 정확도 계산해서 추가

    #확률 표현된 것을 추가한다
    prob_matrix = confusion_matrix[:num_classes, :num_classes] / total_predictions

    return confusion_matrix, prob_matrix

#원핫 함수
def One_Hot(data):
    num_classes = len(np.unique(data[:, -1]))
    one_hot_encoded = np.eye(num_classes)[data[:, -1].astype(int)]
    return one_hot_encoded

#정확도 계산
def accuracy(y_true, y_pred):
    correct_predictions = np.sum(y_true == y_pred)
    accuracy = correct_predictions / len(y_true)
    return accuracy

#mse계산
def mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

#순전파만 진행한 신경망
def Neural_Network(data, w_hidden, w_output):
    x_with_dummy = np.hstack([data[:, :-1], np.ones((data.shape[0], 1))])
    _, _, _, y_hat = forward_propagation(x_with_dummy, w_hidden, w_output)
    y_pred = np.round(y_hat).astype(int)
    return y_hat, y_pred
```

확률을 구하기 위해 confusion_matrix 함수를 수정하고, 정확도, mse함수를 추가, 순전파만 진행하는 Neural_Network 함수를 추가하여 각 함수를 실행하면 가중치를 알 수 있다.

```
train_set, test_set = aug_data(data_set, 0.8, 0.2) # 데이터 분할

One_hot_Test = One_Hot(test_set) # Test 데이터에 대한 One_hot 생성

# 가중치 불러오기
best_v = pd.read_csv('C:\\Users\\user\\OneDrive - 한국공학대 학교\\바탕 화면\\3학년 1학기\\머신러닝실습\\Machine-Learning\\프젝\\weight\\w_hidden0.6 - 1.csv', header=None).to_numpy()
best_w = pd.read_csv('C:\\Users\\user\\OneDrive - 한국공학대 학교\\바탕 화면\\3학년 1학기\\머신러닝실습\\Machine-Learning\\프젝\\weight\\w_output0.6 - 1.csv', header=None).to_numpy()
w_hidden, w_output = best_v, best_w

# Test Set 순전파 진행해서 y_hat을 받아오기
y_hat_Test, Y_hat_Test = Neural_Network(test_set, w_hidden, w_output)

# Test 정확도 저장
Test_Acc = accuracy(test_set[:, -1].reshape(-1, 1), Y_hat_Test)

# Test 평균제곱오차 저장
MSE_Test = mse(test_set[:, -1].reshape(-1, 1), y_hat_Test)

# Confusion Matrix 생성
Confusion_Matrix_Test, Prob_Matrix_Test = compute_confusion_matrix_with_probabilities(test_set[:, -1], Y_hat_Test, num_classes=2)
```


가중치(W, V) 선택

앞서 여러 번 시도한 데이터에 대해 나온 가중치 W,V를 테스트용 코드로 yes, no를 동일하게 선택해 테스트
이 테스트 코드는 순전파만 이용하여 정확도 계산 부분만 실행후 confusion_matrix 분석

	0	1	2
0	373	248	0.600644
1	127	252	0.664908
2	0.746	0.504	0.625

	0	1	2
0	394	248	0.613707
1	106	252	0.703911
2	0.788	0.504	0.646

가중치 구하는 코드에서의 정확도 0.63

	0	1	2
0	379	253	0.599684
1	121	247	0.671196
2	0.758	0.494	0.626

	0	1	2
0	374	253	0.596491
1	126	247	0.662198
2	0.748	0.494	0.621

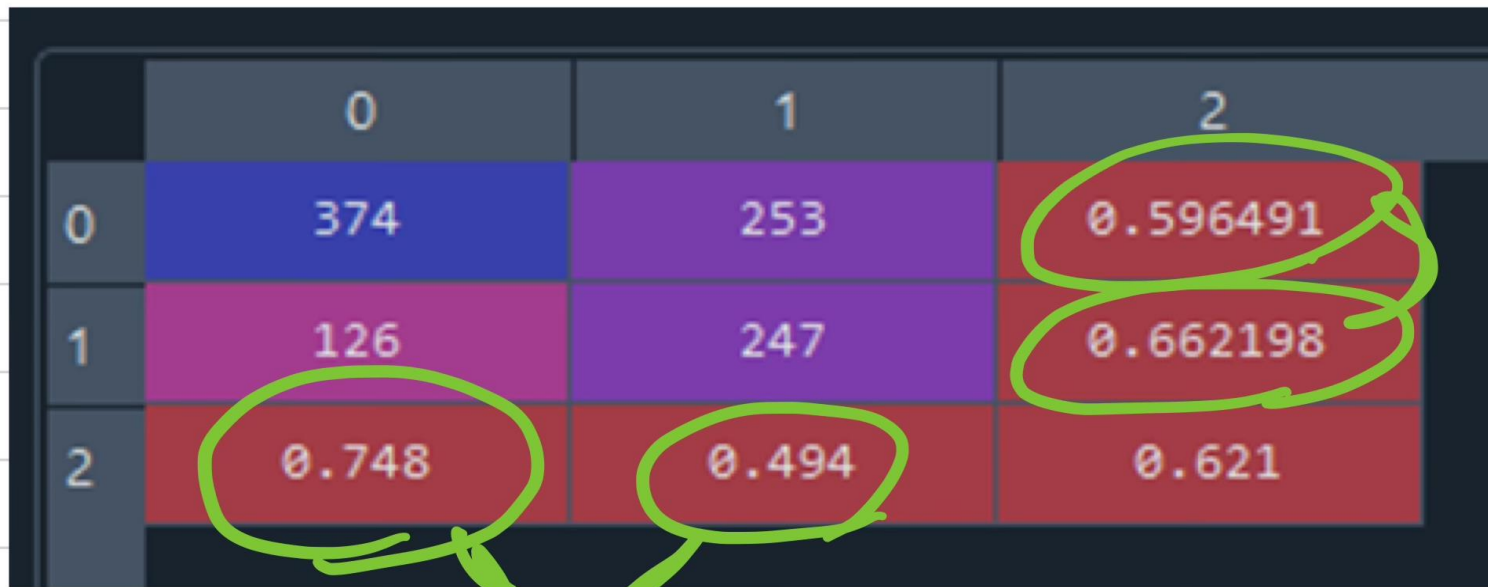
가중치 구하는 코드에서의 정확도 0.65

각 가로별로 같은 코드를 여러 번 실행하였더니, 우측 하단이 정확도이고, 기존 가중치를 구하는 코드에서 처럼
정확도는 비슷하게 나오는 것을 알 수 있다.

0을 0으로 분류는 높게 75퍼가 나오게 되지만, 1을 1로 분류하는 것이 높으면 50퍼로 나오게 된다.

최적의 가중치 선택

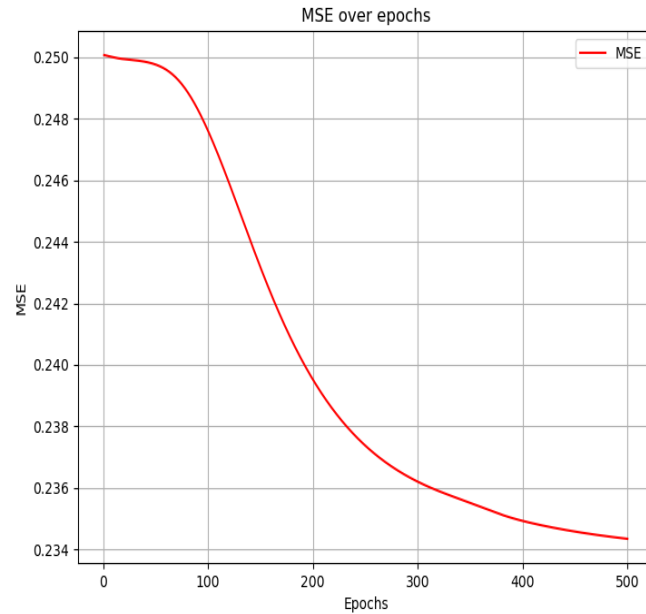
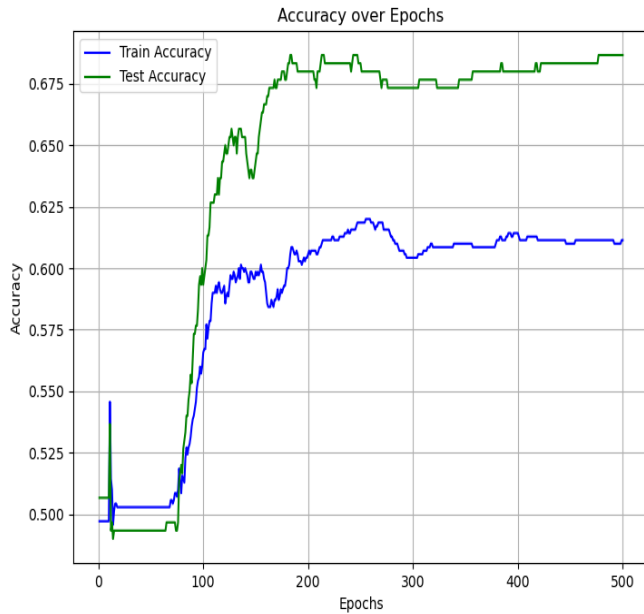
최적의 가중치를 어떤 것을 선택하여야 하는지 생각을 해 보았고, 결국 정확도는 70이 최대치라고 가정하였고, 가중치를 검증하는 코드에서의 confusion_matrix를 실행시키며



	0	1	2
0	374	253	0.596491
1	126	247	0.662198
2	0.748	0.494	0.621

앞서 구한 이 confusion_matrix에서 0을 0으로 옳게 분류하고, 1을 1로 옳게 분류한 확률인 0.748과 0.494의 간격 차이를 줄이면서 가장 차이가 낮은 것을 선택하였다.

최종 가중치와 정확도



best_v - NumPy object array

	0	1	2	3
0	0.0552486	-0.257166	0.00529547	-0.0095911
1	0.0359815	-0.164156	0.00676054	-0.00017876

best_w - NumPy object array

	0	1	2
0	0.360668	0.186735	-0.134461

best_accuracy float64 1 0.686666666666...

confusion_matrix - NumPy object array

	0	1
0	124	66
1	28	82

Confusion_Matrix_Test - NumPy object array

	0	1	2
0	370	223	0.623946
1	130	277	0.68059
2	0.74	0.554	0.647

최종적으로 특징추출에서의 정확도가 최대 68%가 나온 가중치들을 사용하였고, 그 가중치에 대한 Confusion_matrix는 좌측의 array를 갖는다
이 가중치를 추가로 테스트를 하기 위한 코드에 대입하였고 그 중 확률의 차이가 가장 적은 것으로 64%의 정확도 도출