전자공학과 2020142001 곽종근

7주차 로지스틱 회귀
Chap.3 실습

제출일: 2024.05.07.

Chap.3

```python
def aug_data(data, train_ratio, test_ratio):
    # 데이터를 분할하는 것이므로 분할한 것들의 합이 1이 나와야 함
    assert train_ratio + test_ratio == 1

    # 데이터의 총 개수
    total_samples = len(total_data)

    # 각 세트의 크기 계산
    train_size = int(total_samples * train_ratio)

    # 데이터를 랜덤하게 섞음
    np.random.shuffle(total_data)

    # 데이터 분할
    train_set = total_data[:train_size]
    test_set = total_data[train_size:]

    return train_set, test_set
# 위에서 선언한 함수를 이용해 7:3으로 트레인과 테스트 분리
train_set, test_set = aug_data(total_data, 0.7, 0.3)

# 분리된 7에 대한 트레인 셋으로 더미데이터 포함한 새로운 값을 만들어준다
train_x_with_dummy = np.hstack((train_set[:, :2], np.ones((len(train_set), 1))))
train_y = train_set[:, 2].reshape(-1, 1)

# 분리된 3에 대한 테스트 셋으로 더미데이터 포함한 새로운 값을 만들어준다
test_x_with_dummy = np.hstack((test_set[:, :2], np.ones((len(test_set), 1))))
test_y = test_set[:, 2].reshape(-1, 1)

w0_history, w1_history, w2_history, cee_history, accuracy_history, test_accuracy_history = gradient_descent(train_x_with_dummy, train_y, 0.3, 4000)
```

```python
# 시그모이드 함수를 선언
def Sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Cee를 함수로 선언
def Cee(X, y, w):
    m = len(y)
    z = np.dot(X, w)
    p = Sigmoid(z)
    cost = (-1 / m) * np.sum(y * np.log(p) + (1 - y) * np.log(1 - p))
    return cost

# 예측값을 선언 이는 y^과 같음
def predict(X, w):
    z = np.dot(X, w)
    p = Sigmoid(z)
    predictions = np.where(p >= 0.5, 1, 0)
    return predictions
```

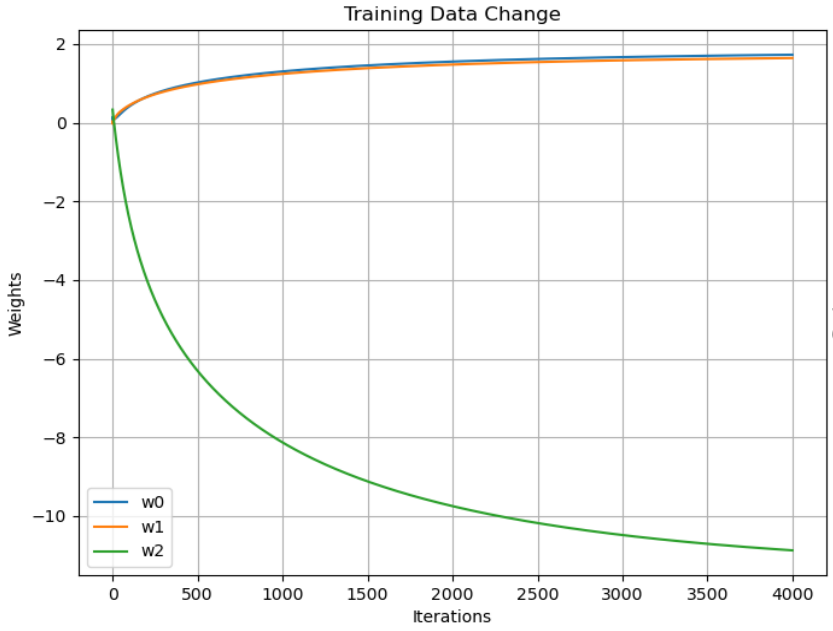
이전 실습에 사용했던 데이터 분할 코드를 이용헤 train과 test로 분리하는 함수를 선언하고 7대3으로 분리
오른쪽 코드는 경사하강법을 사용하기 위해 시그모이드, Cee, 예측값을 따로 함수로 선언

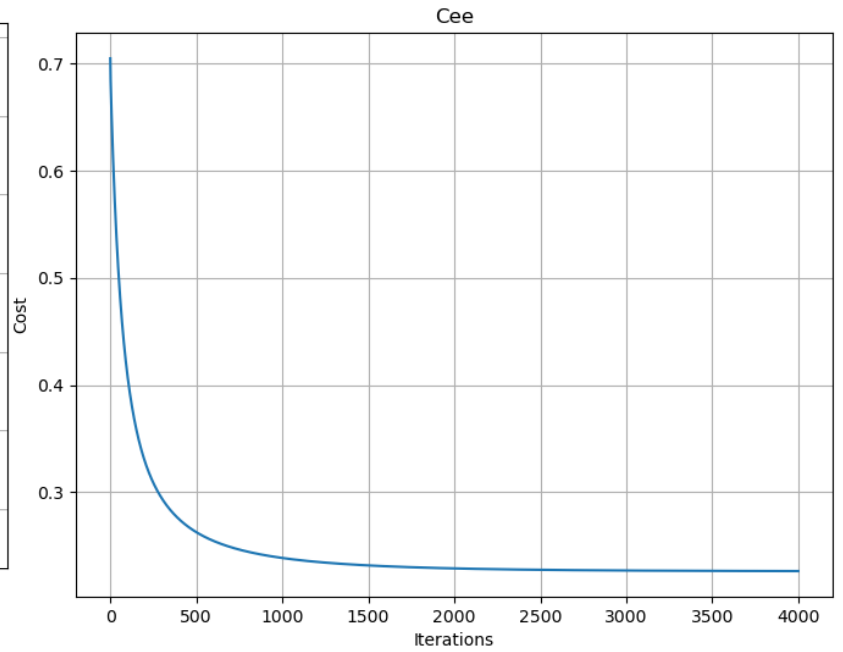
Chap.3

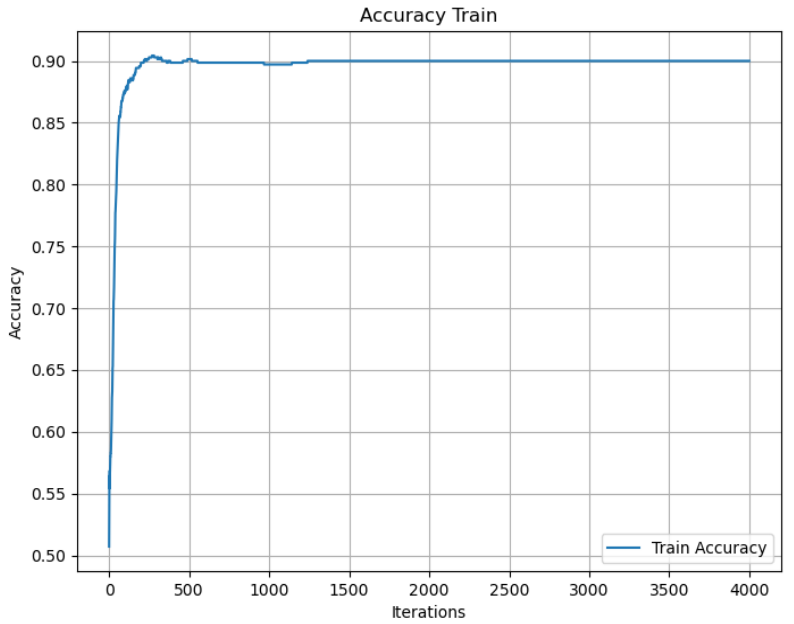
```python
# 위에서 선언한 함수들을 이용해 경사하강법 진행
def gradient_descent(X, y, alpha, rp):
    # 가중치 초기값을 랜덤으로 선언
    w_ = np.random.rand(3, 1)
    w0_history, w1_history, w2_history, cee_history, accuracy_history, test_accuracy_history = [], [], [], [], [], []

    for i in range(rp):
        z = np.dot(X, w_)
        p = Sigmoid(z)
        # axis를 선언해 축별로 계산되도록
        dif_cee = np.mean((p - y) * X, axis=0).reshape(-1, 1)
        w_ -= alpha * dif_cee

        w0_history.append(w_[0][0])
        w1_history.append(w_[1][0])
        w2_history.append(w_[2][0])

        cee = Cee(X, y, w_)
        cee_history.append(cee)

        predictions = predict(X, w_)
        accuracy = np.sum(predictions == y) / len(y)
        accuracy_history.append(accuracy)

        # 테스트 데이터셋에 대한 정확도 계산
        test_predictions = predict(test_x_with_dummy, w_)
        test_accuracy = np.sum(test_predictions == test_y) / len(test_y)
        test_accuracy_history.append(test_accuracy)

    return w0_history, w1_history, w2_history, cee_history, accuracy_history, test_accuracy_history
```

앞 슬라이드에 있는 코드를 사용해 history리스트에 경사하강법을 진행한 값들을 전부 저장

Chap.3

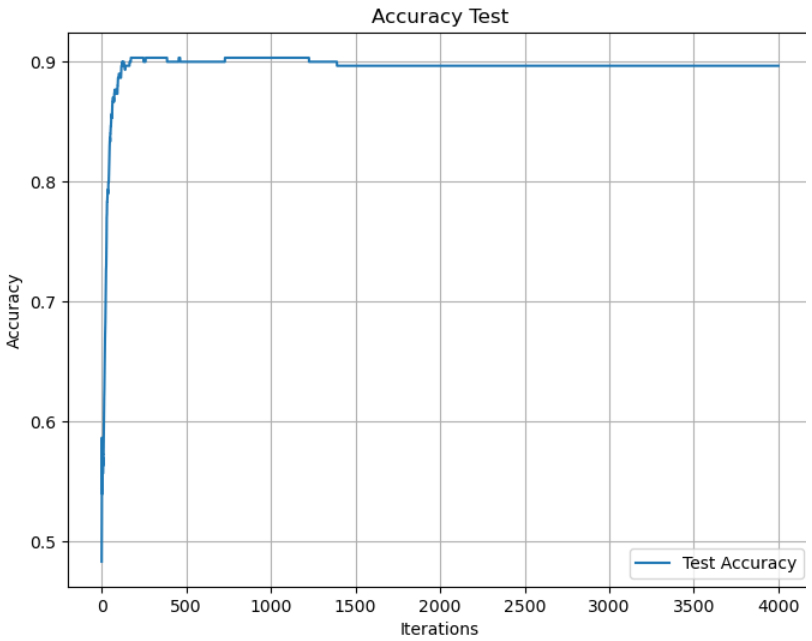


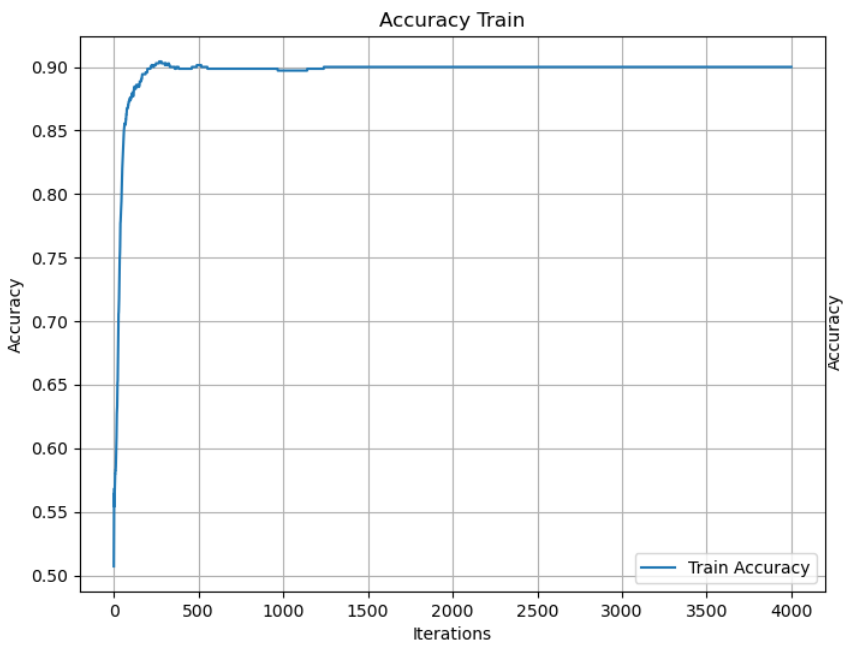


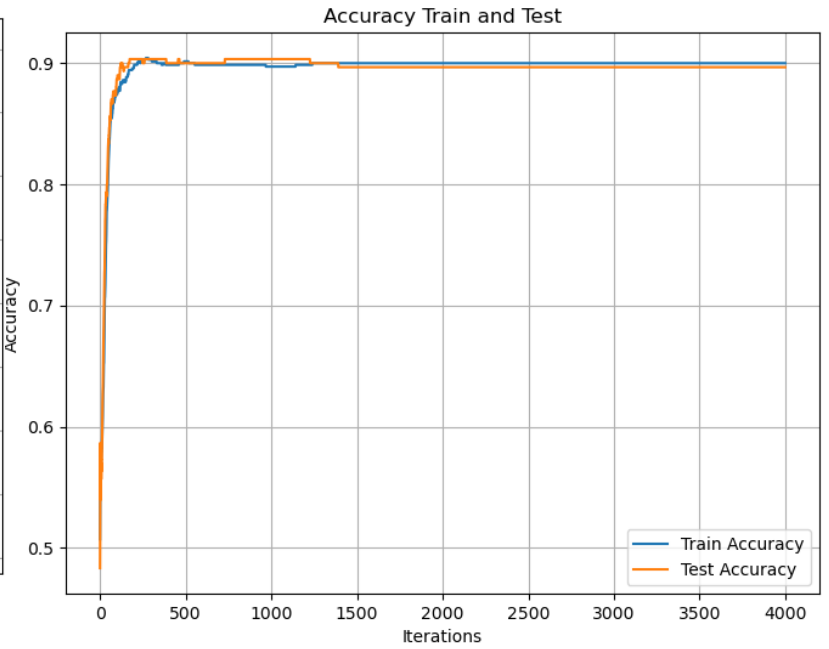


왼쪽 그래프부터 가중치, Cee, 분류 정확도 그래프로 나타나고
가중치 그래프에서 w0과 w1은 매우 근사한 값을 가진다는 것을 알게 되었고
Cee그래프는 시작값에서 급강하여 0과 가까운 값으로 점차 수렴해 가게 되고
정확도 그래프는 100프로 정확하지는 않지만 90프로 정도의 정확도가 나오고 그 이유는
원래 데이터 값을 점찍어 보면 튀어나온 몇몇 값들 때문임을 예상해 볼 수 있다.

Chap.3

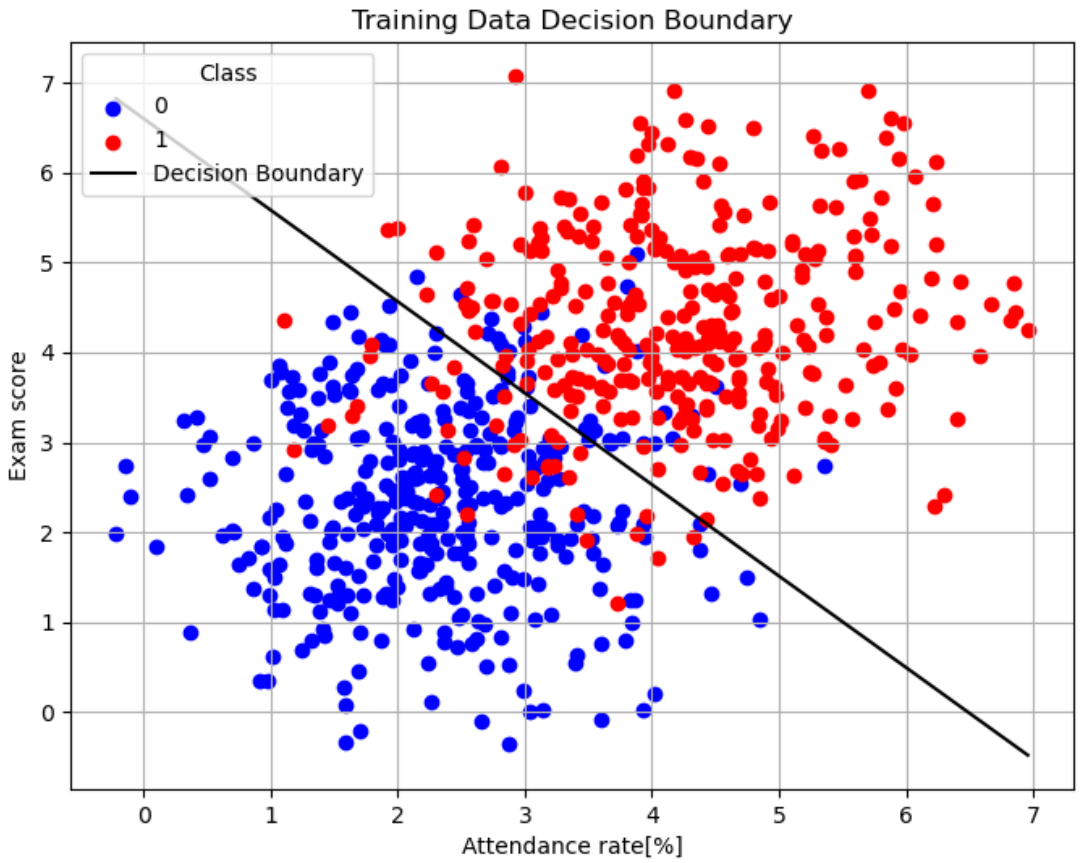


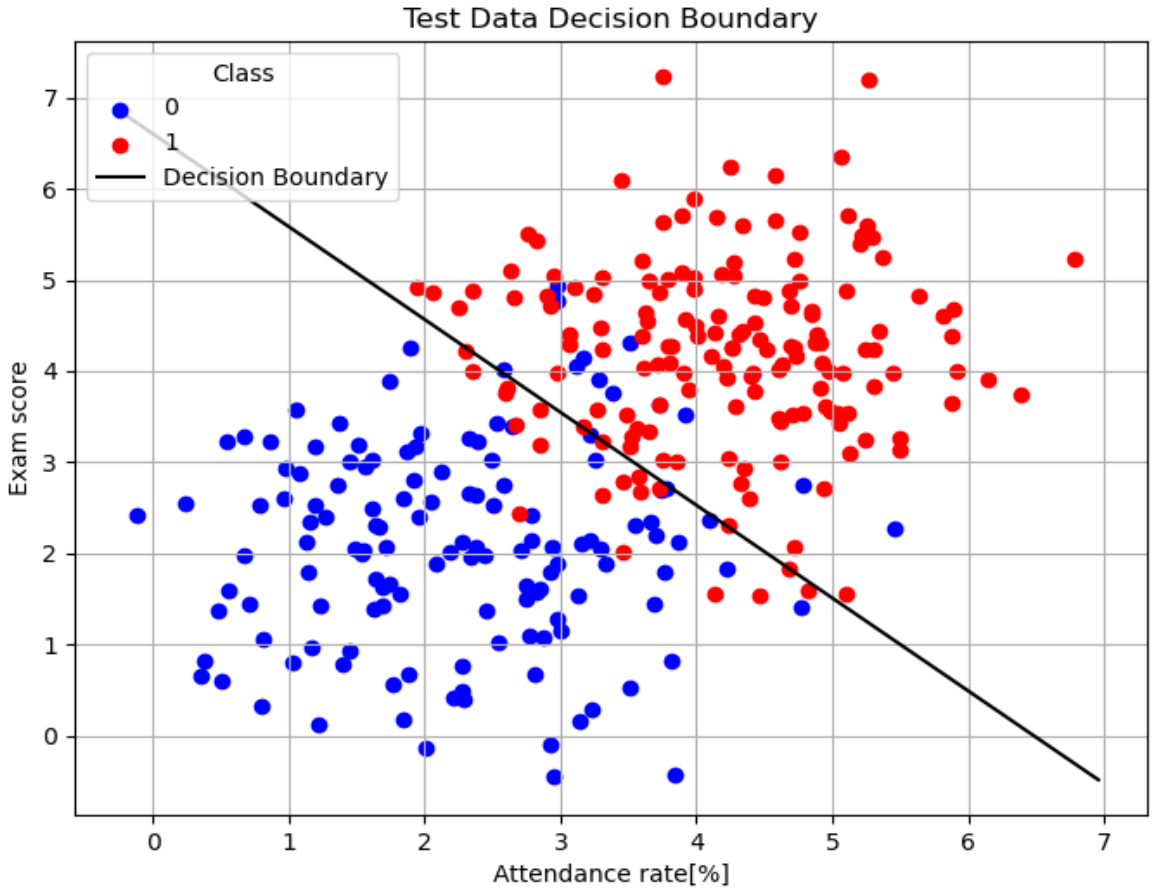




위 그래프는 각각 테스트 정확도, 트레인정확도, 두 정확도 그래프를 하나로 보여준 그래프이고
두 정확도가 모두 90프로에 가까운 높은 정확도를 갖게 되는 것을 알 수 있다.

Chap.3





왼쪽은 train데이터에 대한 결정경계 그래프이고, 오른쪽은 test 데이터에 대한 결정경계 그래프인데
이 그래프를 보면 거의 비슷한 모양이고, scatter 된 값들을 보면 경계를 넘어가있는 값들이 있는데 이들 때문에