2020142001 곽종근
제출일: 2024.08.08.

중간 프로젝트

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 시그모이드 함수
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
# 시그모이드 함수의 미분
def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))
# 순전파 함수
def forward_propagation(x_with_dummy, v, w):
    A = v @ x_with_dummy.T #은닉층
    b = sigmoid(A) #은닉층 출력
    b_with_dummy = np.vstack([b, np.ones([1, len(x_with_dummy)])])
    B = w @ b_with_dummy #출력층
    y_hat = sigmoid(B) #출력층 출력
    return A, b, b_with_dummy, B, y_hat
# 역전파 함수
def backward_propagation(x_with_dummy, y, A, b, b_with_dummy, B, y_hat, v, w):
    error = y_hat - y.T #질제와 예측 차이로 오차 계산
    wmse = (error * sigmoid_derivative(B)) @ b_with_dummy.T / len(x_with_dummy) # 출력층 가중치의
업데이트
    vmse = ((w[:, :-1].T @ (error * sigmoid_derivative(B))) * sigmoid_derivative(A)) @ x_with_dummy /
len(x_with_dummy) #은닉층 가중치의 업데이트
    return wmse, vmse

# 데이터셋을 학습셋과 테스트셋으로 분할
def aug_data(data, train_ratio, test_ratio):
    assert train_ratio + test_ratio == 1 # 테스트와 트레인 비율의 합은 1
    total_samples = len(data)
    train_size = int(total_samples * train_ratio)
    np.random.shuffle(data)
    train_set = data[:train_size]
    test_set = data[train_size:]
    return train_set, test_set

# confusion matrix 함수
def compute_confusion_matrix(y_true, y_pred, num_classes):
    confusion_matrix = np.zeros((num_classes, num_classes))
```

```
    for i in range(len(y_true)):
        row_index = int(y_pred[i]) #예측값
        col_index = int(y_true[i]) #실제값
        confusion_matrix[row_index, col_index] += 1
    return confusion_matrix

# 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 저장하여
    # 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)

    # 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 이상한 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
```

```
directory = "C:\\Users\\user\\OneDrive - 한국공학대학교\\바탕 화면\\3학년
1학기\\머신러닝실습\\Machine-Learning\\"
features = select_features(directory)


#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 민감하게 반응
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

# confusion matrix 계산
confusion_matrix = compute_confusion_matrix(y_test.flatten(), y_hat_test_index, 2)

print("Confusion Matrix:")
print(confusion_matrix)

# 그래프 출력
plt.figure(figsize=(18, 6))

plt.subplot(1, 2, 1)
plt.plot(range(1, epochs+1), accuracy_list, label='Train Accuracy', color='blue')
plt.plot(range(1, epochs+1), test_accuracy_list, label='Test Accuracy', color='green')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy over Epochs')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(range(1, epochs+1), mse_list, label='MSE', color='red')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.title('MSE over epochs')
plt.legend()
plt.grid()


plt.show()

best_v_save = pd.DataFrame(best_v)
best_w_save = pd.DataFrame(best_w)
best_v_save.to_csv('C:\\Users\\user\\OneDrive - 한국공학대학교\\바탕 화면\\3학년
1학기\\머신러닝실습\\Machine-Learning\\프젝\\weight\\w_hidden0.6 - 3.csv', header=None, index=False)
best_w_save.to_csv('C:\\Users\\user\\OneDrive - 한국공학대학교\\바탕 화면\\3학년
1학기\\머신러닝실습\\Machine-Learning\\프젝\\weight\\w_output0.6 - 3.csv', header=None, index=False)

# best_v = pd.read_csv('C:\\Users\\user\\OneDrive - 한국공학대학교\\바탕 화면\\3학년
1학기\\머신러닝실습\\Machine-Learning\\프젝\\w_hidden0.61.csv', header=None)
# best_w = pd.read_csv('C:\\Users\\user\\OneDrive - 한국공학대학교\\바탕 화면\\3학년
1학기\\머신러닝실습\\Machine-Learning\\프젝\\w_output0.61.csv', header=None)
```