7주차 실습과제 코드

2020142001 곽종근
제출일: 2024.05.02.

실습과제 4.

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

# 데이터 불러오기
fold_dir = "C:\\Users\\user\\OneDrive - 한국공학대학교\\바탕 화면\\3학년
1학기\\머신러닝실습\\Machine-Learning\\5주차\\lin_regression_data_01.csv"
temp_data = pd.read_csv(fold_dir, header=None)
temp_data = temp_data.to_numpy()     # 데이터를 numpy로 변환

# 데이터 분리
x_data = temp_data[:, 0]  # 무게 데이터를 Wei저장
y_data = temp_data[:, 1]  # 길이 데이터를 Len에 저장

# 가우시안 기저 함수 정의
def gaussian_basis_function(X, K, k):
    x_min = X.min()  # 데이터의 최솟값
    x_max = X.max()  # 데이터의 최댓값
    mu = x_min + ((x_max - x_min) / (K - 1)) * k  # 각 가우시안 함수의 평균 계산

    v = (x_max - x_min) / (K - 1)  # 모든 가우스 함수의 분산

    simple = (X - mu) / v
    G = np.exp((-1/2) * (simple ** 2))

    return G

# 가중치 계산 함수
def calculate_weights(X, Y, K):
    # k의 배열 생성
    k_values = np.arange(K).reshape(-1, 1)
    # K에 따른 가우시안 기저 함수 계산
    X_b = np.column_stack([gaussian_basis_function(X, K, k) for k in k_values])
    # bias 추가
    X_b = np.hstack([X_b, np.ones((len(X), 1))])
    # 가중치 계산 (K+1개의 가중치)
    weights = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ Y
    return weights

# MSE 계산 함수
```

```
def mse(X, Y, K):
    # k의 배열 생성
    k_values = np.arange(K).reshape(-1, 1)
    # K에 따른 가우시안 기저 함수 계산
    X_b = np.column_stack([gaussian_basis_function(X, K, k) for k in k_values])
    # bias 추가
    X_b = np.hstack([X_b, np.ones((len(X), 1))])
    # 가중치 계산
    weights = calculate_weights(X, Y, K)
    # MSE 계산
    mse_value = np.mean(((X_b @ weights) - Y) ** 2)
    return mse_value

# K_values에 따른 MSE 계산
K_values = np.arange(3, 11)  # K 값 범위 설정
mse_values = [mse(x_data, y_data, K) for K in K_values]

# MSE 그래프 그리기
plt.figure(figsize=(10, 6))
plt.plot(K_values, mse_values, marker='o')
plt.xlabel('Number of Basis Functions (K)')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('MSE vs. Number of Basis Functions')
plt.grid(True)
plt.show()


# 가중치 계산
K_values = [10]
weights_list = [calculate_weights(x_data, y_data, K) for K in K_values]

# 그래프 그리기
plt.figure(figsize=(10, 6))
plt.scatter(x_data, y_data, color='blue', label='Original Data')  # 원본 데이터
for K, weights in zip(K_values, weights_list):
    x_range = np.linspace(x_data.min(), x_data.max(), 1000)
    # bias 추가하여 예측값 계산
    y_pred = np.column_stack([gaussian_basis_function(x_range, K, k) for k in range(K)]) @ weights[:-1] +
weights[-1]
    plt.plot(x_range, y_pred, label=f'Regression Curve (K={K})')  # 회귀 곡선 그리기
plt.xlabel('Weight')
plt.ylabel('Length')
plt.title('Regression Curves with Different K')
plt.legend()
plt.grid(True)
plt.show()
```

과제 chap.2

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

# 데이터 불러오기
fold_dir = "C:\\Users\\user\\OneDrive - 한국공학대학교\\바탕 화면\\3학년
1학기\\머신러닝실습\\Machine-Learning\\5주차\\lin_regression_data_01.csv"
temp_data = pd.read_csv(fold_dir, header=None)
temp_data = temp_data.to_numpy()

# 데이터 분리
Wei = temp_data[:, 0]  # 무게 데이터
Len = temp_data[:, 1]  # 길이 데이터

# 그래프 그리기
plt.figure(figsize=(10, 6))

noise=0.3
# 데이터 증강
augmented_Wei = []
augmented_Len = []
for i in range(len(Wei)):
    for _ in range(20):
        #노이즈를 위에서 변수로 선언하여 위의 값만 바꿔주면 +-노이즈만큼 노이즈가 설정된다.
        augmented_Wei.append(Wei[i] + np.random.rand()*2*noise-noise)

        augmented_Len.append(Len[i] + np.random.rand()*2*noise-noise)

# 증강된 데이터를 numpy 배열로 변환
augmented_data = np.column_stack((augmented_Wei, augmented_Len))

# 분할하고자 하는 입력 비율을 입력받을수있음
# train = float(input("train_set비율: "))
# val = float(input("val_set비율: "))
# test= float(input("test_set비율: "))

def aug_data(augmented_data, train_ratio, val_ratio, test_ratio):

    # 데이터를 분할하는 것이므로 분할한 것들의 합이 1이 나와야함
    assert train_ratio + val_ratio + test_ratio == 1

    # 데이터의 총 개수
    total_samples = len(augmented_data)
```

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

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

    # 데이터 분할
    train_set = augmented_data[:train_size]
    val_set = augmented_data[train_size:train_size + val_size]
    test_set = augmented_data[train_size + val_size:]

    return train_set, val_set, test_set

# 데이터 증강된 것을 먼저 그래프에 추가
plt.scatter(augmented_Wei, augmented_Len, color='red', alpha=0.5, marker='o', s=20, label='Augmented
Data')

# 원본 데이터 그리기
plt.scatter(Wei, Len, color='blue', label='Original Data', s=30)

plt.xlabel('Weight')
plt.ylabel('Length')
plt.title('Original Data & Augmented Data')
plt.legend()
plt.grid(True)
plt.show()

# 데이터 분할
train_set, val_set, test_set = aug_data(augmented_data, 0.8, 0, 0.2)

# 가우시안 기저 함수 정의
def gaussian_basis_function(X, K, k):
    x_min = X.min()  # 데이터의 최솟값
    x_max = X.max()  # 데이터의 최댓값
    mu = x_min + ((x_max - x_min) / (K - 1)) * k  # 각 가우시안 함수의 평균 계산

    v = (x_max - x_min) / (K - 1)  # 모든 가우스 함수의 분산

    simple = (X - mu) / v  #간단하게 표현하기 위해서 사용
    G = np.exp((-1/2) * (simple ** 2))

    return G

# 가중치 계산 함수
def calculate_weights(X_train, Y_train, K):
    k_values = np.arange(K).reshape(-1, 1)
```

```
    X_b = np.column_stack([gaussian_basis_function(X_train, K, k) for k in k_values])
    X_b = np.hstack([X_b, np.ones((len(X_train), 1))]) #1인 더미 데이터를 포함해서 합쳐준다
    weights = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ Y_train
    return weights

# MSE 계산 함수, 가중치를 train으로 해야하기에 입력을 따로 받아준다.
def mse(X, X_train, Y, Y_train, K):
    k_values = np.arange(K).reshape(-1, 1)
    X_b = np.column_stack([gaussian_basis_function(X, K, k) for k in k_values])
    X_b = np.hstack([X_b, np.ones((len(X), 1))])
    weights = calculate_weights(X_train, Y_train, K)
    mse_value = np.mean(((X_b @ weights) - Y) ** 2)
    return mse_value

## 다양한 K에 대한 MSE 그래프 그리기
K_values = range(2, 70)  # K 값 범위 설정
train_mse_values = []  # 훈련 데이터에 대한 MSE 저장 리스트
test_mse_values = []   # 테스트 데이터에 대한 MSE 저장 리스트

for K in K_values:
    # 훈련 데이터에 대한 MSE 계산 및 저장
    train_mse = mse(train_set[:, 0], train_set[:, 0], train_set[:, 1], train_set[:, 1], K)
    train_mse_values.append(train_mse)

    # 테스트 데이터에 대한 MSE 계산 및 저장, 이때 가중치를 train_set로 해주기 위해 x값을 받는 것 추가
    test_mse = mse(test_set[:, 0], train_set[:, 0], test_set[:, 1], train_set[:, 1], K)
    test_mse_values.append(test_mse)

# 최적의 K 값 찾기
optimum_k = np.argmin(test_mse_values)
print("최적의 K 값:", K_values[optimum_k])

# 그래프 그리기
plt.figure(figsize=(10, 6))
plt.plot(K_values, train_mse_values, label='Train MSE')
plt.plot(K_values, test_mse_values, label='Test MSE')
plt.xlabel('K')
plt.ylabel('MSE')
plt.title('MSE & K')
plt.legend()
plt.grid(True)
plt.show()
```