1주차 (Chapter 1,2)

1주차 과제: 데이터 스케일링의 영향 분석 (상세 가이드)

1. 과제의 목표

이 과제의 핵심 목표는 **왜(Why)** 데이터 전처리가 중요한지, 특히 특성(feature)들의 스케일이 다를 때 모델(특히 k-NN)이 어떻게 오작동할 수 있는지, 그리고 **스케일링**이라는 간단한 처리가 어떻게 문제를 해결하는지를 코드로 직접 확인하고 체감하는 것입니다.

2. 실험 과정 (Step-by-Step)

교재의 `02-1` 절 코드를 기반으로 진행합니다. 생선의 `길이(length)`와 `무게(weight)`라는 두 가지 특성을 사용해 도미와 빙어를 분류하는 예제입니다.

Step 1: 스케일링 *전* 데이터로 모델 훈련 및 평가

1. **데이터 준비:** 교재의 코드처럼 도미(bream)와 빙어(smelt) 데이터를 합쳐 `fish_data` (입력 데이터)와 `fish_target` (정답 데이터)를 만듭니다.Python

```
# 예시 코드 (교재 내용)
fish_data = [[25.4, 242.0], [26.3, 290.0], ... , [10.0, 9.8]]
fish_target = [1, 1, ... , 0, 0]
```

2. **훈련/테스트 세트 분리:** `train_test_split()` 함수를 사용하거나, 교재처럼 인덱싱을 통해 훈련 세트와 테스트 세트를 나눕니다.Python

```
# 예시 코드 (교재 내용)
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(
    fish_data, fish_target, stratify=fish_target, random_state=42)
```

3. **모델 훈련 및 평가:** **아무런 전처리를 하지 않은 원본 데이터 (`train_input`, `test_input`)**를 사용해 k-최근접 이웃 모델을 훈련시키고 정확도를 측정합니다.Python

```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier()
kn.fit(train_input, train_target)
```

```
score_before = kn.score(test_input, test_target)

print(f"스케일링 전 정확도: {score_before}")
# 아마 1.0 또는 매우 높은 점수가 나올 것입니다.
```

4. **문제점 확인 (가장 중요한 부분!):** 정확도가 1.0이 나왔다고 해서 모델이 완벽한 것은 아닙니다. 교재의 예시처럼, 특정 샘플(예: `[25, 150]`)을 예측해보면 잘못된 결과가 나옵니다.
    - **이유 분석:** 산점도(scatter plot)를 그려보면 명확히 알 수 있습니다.
        - x축(길이)의 범위: 약 10 ~ 40
        - y축(무게)의 범위: 약 10 ~ 1000
        - y축의 범위가 x축보다 훨씬 크기 때문에, k-NN이 두 점 사이의 거리를 계산할 때 **'무게' 특성이 거리 계산을 거의 독점**하게 됩니다. '길이' 특성은 거의 무시되는 셈이죠. 이것이 바로 스케일링이 필요한 이유입니다.

Step 2: 스케일링 *후* 데이터로 모델 훈련 및 평가

1. **데이터 스케일링 (표준화):** `StandardScaler`를 사용해 훈련 세트의 평균과 표준편차를 계산하고, 이를 이용해 훈련 세트와 테스트 세트를 모두 변환(스케일링)합니다.Python
    - **주의!** `fit()`은 반드시 **훈련 세트(`train_input`)에만** 적용해야 합니다. 훈련 세트에서 학습한 통계값(평균, 표준편차)을 그대로 테스트 세트에 적용(`transform()`)해야 합니다.

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(train_input) # 훈련 세트에서만 통계값 계산

# 훈련 세트와 테스트 세트 모두 변환
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```

2. **모델 훈련 및 평가:** 이제 **스케일링된 데이터(`train_scaled`, `test_scaled`)**를 사용해 k-NN 모델을 다시 훈련하고 평가합니다.Python

```
kn.fit(train_scaled, train_target)
score_after = kn.score(test_scaled, test_target)
```

```
print(f"스케일링 후 정확도: {score_after}")
# 여전히 1.0이 나올 수 있지만, 이제 모델은 각 특성을 공정하게 바라봅니다.
```

3. **결과 확인:** 이전에 잘못 예측했던 샘플(예: `[25, 150]`)도 **동일하게 스케일링**한 후 예측을 수행해봅니다.Python

```
new_sample = [[25, 150]]
scaled_sample = ss.transform(new_sample)

print(kn.predict(scaled_sample))
# 이제 올바른 값(도미, 즉 '1')으로 예측하는 것을 확인할 수 있습니다.
```

3. 과제 결과 정리 및 공유 (노션, 깃허브 등)

스터디원들과 공유할 결과물은 아래와 같은 내용을 포함하여 정리하면 좋습니다.

1. **실험 제목:** 데이터 스케일링이 k-NN 모델 성능에 미치는 영향 분석
2. **실험 결과 요약:**
   - 스케일링 전 정확도: `1.0`
   - 스케일링 후 정확도: `1.0`
   - 특정 샘플 `[25, 150]` 예측 결과:
     - 스케일링 전: `0` (빙어) - **오답**
     - 스케일링 후: `1` (도미) - **정답**
3. **결론 및 분석 (가장 중요):**
   - **왜 이런 결과가 나왔는가?**
     - "초기 데이터는 생선의 '길이'(10~40)와 '무게'(10~1000) 특성의 값 범위(스케일) 차이가 매우 크다."
     - "k-NN과 같은 거리 기반 알고리즘은 이 스케일 차이 때문에 값이 큰 '무게' 특성에만 크게 의존하여 거리를 계산하게 된다."
     - "이로 인해 스케일링 전 모델은 정확도는 1.0이었지만, 실제로는 '무게'만 보고 판단하는 편향된 모델이었으며, 특정 샘플에 대해 오답을 내었다."
   - **스케일링의 역할과 효과:**
     - "표준화(Standardization)를 통해 모든 특성의 평균을 0, 표준편차를 1로 맞춰주자, 두 특성이 동등한 조건에서 거리에 반영되었다."

    - "그 결과, 모델이 더 안정적으로 올바른 예측을 할 수 있게 되었다."
- **최종 결론:**
    - "따라서, 특성 간 스케일 차이가 큰 데이터를 k-NN과 같은 거리 기반 모델에 사용할 때는 **데이터 스케일링이 필수적인 전처리 과정**임을 확인했다."

4. **(Optional) 시각화 자료:**
    - 스케일링 전/후 데이터의 산점도를 함께 첨부하면 스케일 축이 어떻게 변했는지 명확하게 보여줄 수 있어 이해도를 크게 높일 수 있습니다.