**K-최근접 이웃(KNN) 알고리즘 분석**

30501 김도현

## **1페이지**

서론 - 예측 모델의 초석, 데이터 분할  
 본 보고서는 K-최근접 이웃(KNN) 알고리즘의 원리를 탐구하고 이를 파이썬 코드로 구현하는 과정을 상세히 기술한다. 기계학습 모델의 성능을 객관적으로 평가하기 위한 가장 기본적인 절차는 보유한 데이터를 학습(train)용과 테스트(test)용으로 분리하는 것이다. 모델이 학습 과정에서 보지 못했던 새로운 데이터에 대해 얼마나 일반화된 예측 성능을 보이는지 확인해야만 실용적인 가치를 지니기 때문이다. 이 노트북에서는 scikit-learn 라이브러리의 train\_test\_split 함수를 활용하여, 주어진 전체 데이터를 학습 데이터 80%, 테스트 데이터 20%의 비율로 무작위 분할하는 과정을 시연했다. 이 과정은 모델의 과적합(overfitting)을 방지하고 일반화 성능을 측정하는 데 필수적이다.

## **2페이지**

분석 대상 - 붓꽃(Iris) 데이터셋의 구조  
 분석에는 기계학습 분류 문제의 'Hello, World'와 같은 위상을 지닌 붓꽃(Iris) 데이터셋을 사용했다. scikit-learn의 datasets.load\_iris()를 통해 로드된 이 데이터셋은 딕셔너리와 유사한 Bunch 객체 형태를 띤다. 주요 키(key)로는 꽃의 특성 데이터(feature)를 담고 있는 data, 각 데이터의 품종(class)을 나타내는 정수 레이블을 담은 target, 그리고 각 특성과 레이블의 실제 이름을 담은 feature\_names와 target\_names가 있다. 코드에서는 입력 데이터 X에 iris\_data.data를, 정답 데이터 Y에 iris\_data.target을 할당함으로써 모델 학습에 필요한 데이터 구조를 명확히 정의했다. 이후 X\_train.shape 등을 통해 분할된 데이터의 차원을 확인함으로써 데이터 준비가 올바르게 완료되었음을 검증했다.

## **3페이지**

데이터의 시각적 탐색과 KNN의 가능성  
 데이터의 특성을 직관적으로 파악하기 위해 matplotlib을 이용한 시각화를 수행했다. 붓꽃 데이터의 4가지 특성(꽃받침 길이/너비, 꽃잎 길이/너비) 중, 품종 간 변별력이 높다고 알려진 3번째(꽃잎 길이)와 4번째(꽃잎 너비) 특성을 각각 x축과 y축으로 설정하여 2차원 산점도(scatter plot)를 그렸다. 각 품종(setosa, versicolor, virginica)은 서로 다른 색상으로 표시되었으며, 그 결과 setosa 품종이 다른 두 품종과 명확하게 분리되어 군집을 이루고 있음을 확인할 수 있었다. Versicolor와 virginica는 일부 겹치는 구간이 존재하지만 대체로 잘 구분되는 양상을 보였다. 이러한 시각적 분포는 '가까운 데이터끼리는 같은 종류일 것이다'라는 KNN 알고리즘의 기본 가정이 이 데이터셋에서 매우 효과적으로 작동할 것임을 강력하게 시사한다.

## **4페이지**

KNN 알고리즘의 원리 - 거리, 이웃, 그리고 다수결  
 KNN은 가장 직관적인 머신러닝 알고리즘 중 하나로, '유유상종(類類相從)'이라는 개념에 기반한다. 예측하려는 새로운 데이터 포인트가 주어지면, 기존 학습 데이터셋의 모든 데이터 포인트와의 거리를 각각 계산한다. 이때 거리 측정 방식으로는 주로 유클리드 거리(Euclidean distance)가 사용된다. 거리를 모두 계산한 후, 새로운 데이터 포인트에서 가장 가까운 순서대로 K개의 '이웃'을 선택한다. 마지막으로, 이 K개의 이웃들이 속한 클래스(품종)를 확인하고, 가장 많은 이웃이 속한 클래스를 새로운 데이터의 예측값으로 결정하는 '다수결(majority voting)' 원칙을 따른다. K값은 사용자가 직접 정해야 하는 하이퍼파라미터(hyperparameter)로, K값에 따라 모델의 결정 경계(decision boundary)가 달라지며 성능에 큰 영향을 미친다.

## **5페이지**

순수 파이썬을 이용한 KNN 클래스 설계 및 구현 (1)  
 scikit-learn과 같은 라이브러리에 의존하지 않고 KNN 알고리즘의 내부 동작을 이해하기 위해, 파이썬 클래스(KNN)를 직접 정의하고 구현했다. 먼저, 두 데이터 포인트 간의 거리를 계산하는 distance 메소드를 유클리드 거리 공식에 따라 구현했다. 클래스의 fit 메소드는 학습 데이터를 받아 내부 변수(self.X\_train, self.Y\_train)에 저장하는 단순한 역할을 수행한다. KNN은 데이터를 저장하는 것 외에 별도의 복잡한 학습 과정을 거치지 않으므로 '게으른 학습(Lazy Learning)' 알고리즘으로 분류되기도 한다. predict 메소드는 테스트 데이터셋 전체(X\_test)를 입력받아 각 데이터 포인트에 대해 개별 예측을 수행하는 \_prediction 메소드를 반복 호출하고, 그 결과를 리스트로 취합하여 반환하는 구조로 설계되었다.

## **6페이지**

KNN 예측 로직의 핵심 구현 (2)  
 예측의 핵심 로직은 \_prediction 메소드에 담겨있다. 하나의 테스트 데이터 포인트(x\_test)가 주어지면, fit 메소드를 통해 저장해 둔 모든 학습 데이터(self.X\_train)와의 유클리드 거리를 계산하여 리스트(distances)에 저장한다. 그 후, 이 거리 리스트를 오름차순으로 정렬하여 가장 거리가 가까운 상위 K개의 데이터 인덱스를 추출한다. 이 인덱스를 사용하여 self.Y\_train에서 해당 이웃들의 실제 클래스 레이블을 가져온다. 마지막으로 collections.Counter를 사용하여 K개의 레이블 중 가장 빈도가 높은, 즉 다수를 차지하는 레이블을 최종 예측 결과로 반환한다. 이렇게 구현된 모델의 성능은 accuracy 함수를 통해 실제 정답(Y\_test)과 예측값(predictions)을 비교하여 정확도로 측정했다.

## **7페이지**

결론 - 구현을 통한 알고리즘의 깊이 있는 이해  
 본 KNN.ipynb는 데이터 준비, 시각화, 알고리즘 원리 설명, 그리고 직접 구현에 이르는 체계적인 과정을 통해 KNN 알고리즘에 대한 깊이 있는 이해를 제공했다. 특히 라이브러리 없이 KNN 클래스를 직접 구현하는 과정은 알고리즘의 내부 동작 메커니즘을 명확히 파악하는 데 결정적인 역할을 했다. 붓꽃 데이터셋에서 100%라는 높은 정확도를 달성했지만, 이는 붓꽃 데이터의 분포가 매우 명확하기 때문임을 인지해야 한다. 실제 복잡한 데이터에서는 최적의 K값 탐색, 데이터 스케일링, 적절한 거리 측정 방식 선택 등 추가적인 고려사항이 모델 성능에 큰 영향을 미친다는 점을 유념해야 할 것이다.