AI Programming

데이터 마이닝 / 생성형 AI

04. 머신러닝 기초 - 지도학습

first coding

first coding

머신러닝 개요 및 지도학습 소개

머신러닝의 정의와 활용 분야

지도학습의 개념 및 지도학습 vs 비지도학습

데이터의 역할과 필요성

머신러닝의 정의와 활용 분야

- 머신러닝
  - 머신러닝은 데이터를 기반으로 스스로 학습하여 패턴을 발견하고,
  - 이를 이용해 미래의 결과를 예측하거나 의사결정을 내리는 기술
  - "규칙(함수)을 사람이 직접 입력하는 대신, 데이터를 통해 컴퓨터가 규칙을 스스로 찾는 과정“
  - 전통적인 프로그래밍 방식에서는 사람이 명확한 규칙(함수)을 정의하고 만들어야 하지만
  - 머신러닝에서는 컴퓨터가 데이터를 기반으로 학습을 통해 규칙(함수)을 만듦

지도학습의 개념 및 지도학습 vs 비지도학습

- **지도학습의 개념**

  - 지도학습(Supervised Learning)
    - 데이터와 함께 해당 데이터에 대한 정답(라벨)을 제공하여 모델이 학습하도록 하는 머신러닝 방법
      → "질문과 답을 주고 학습시키는 방식"
    - 모델은 데이터와 정답의 관계를 학습하고, 이후에는 새로운 데이터를 입력 받았을 때 정답을 예측

    학생과 교사의 관계

    → 교사가 문제(데이터)와 정답(라벨)을 함께 제공하며 학생이 문제를 푸는 방법을 배움.

    예: "사과와 바나나 사진을 보여주고, 각각의 이름을 가르쳐주는 과정."

지도학습의 개념 및 지도학습 vs 비지도학습

- **비지도학습의 개념**

  - 비지도학습(Unsupervised Learning)
    - 데이터에는 있지만 정답(라벨)은 주어지지 않은 상태에서 학습하는 머신러닝 방법
      → "질문만 주고, 답은 스스로 찾도록 하는 방식"
    - 모델은 데이터를 스스로 분석해 패턴이나 그룹을 찾아냄

    탐구 학습

    → 학생이 문제(데이터)를 스스로 탐구하며 규칙과 패턴을 찾아냄.

    예: "사과와 바나나 사진을 보여주고, 스스로 사과와 바나나의 차이점을 발견하는 과정."

지도학습의 개념 및 지도학습 vs 비지도학습

- 지도학습 vs 비지도학습

특징	지도학습	비지도학습
라벨(정답) 제공 여부	데이터에 정답(라벨)이 있음	정답(라벨)이 없음
학습 목적	입력 데이터와 정답 사이의 관계를 학습	데이터의 패턴이나 그룹을 학습
사례	- 스팸 메일 분류 - 집값 예측 - 이미지 분류	- 고객 그룹화 - 추천 시스템 - 데이터 압축
적용 분야	예측 및 분류 작업	데이터 이해 및 탐색

지도학습의 개념 및 지도학습 vs 비지도학습

- 머신러닝

  - 입력(Features)와 출력(Target)의 개념
    - 입력(Features)
      - 모델이 학습할 때 사용하는 데이터의 특성
      - 데이터를 구성하는 각각의 요소로, 예측이나 분류에 영향을 미치는 중요한 정보
    - 출력(Target)
      - 모델이 예측해야 하는 결과 값으로, 문제의 정답이거나 우리가 구하고자 하는 목표

지도학습의 개념 및 지도학습 vs 비지도학습

- 머신러닝

  - 집값 예측문제 : 집의 크기, 방 개수, 위치 등을 바탕으로 집값을 예측하고 싶음.
    - 입력(Features)
      - 집의 크기 (평수)
      - 방의 개수
      - 위치 (도심, 외곽 등)
      - 건축 연도
    - 출력(Target) : 집의 예상 가격

- 모델은 입력 데이터를 바탕으로 집값(출력)을 예측하도록 학습
  예를 들어,
  방이 많고 도심에 가까운 집은 가격이 높을 가능성이 있음

지도학습의 개념 및 지도학습 vs 비지도학습

- 머신러닝

  - 스팸 이메일 분류 문제 : 이메일이 스팸인지 아닌지를 분류하고 싶음.
    - 입력(Features)
      - 이메일 제목에 포함된 단어들
      - 이메일 본문 길이
      - 발신자의 도메인 (예: "gmail.com", "unknown.com")
    - 출력(Target) : "스팸" 또는 "스팸 아님"

- 특정 단어(예: "무료", "당첨")가 많이 포함된 이메일은 스팸일 가능성이 높음
- 발신자가 잘 알려지지 않은 도메인일 경우 스팸일 가능성이 높음

지도학습의 개념 및 지도학습 vs 비지도학습

- 머신러닝

  - 지도학습의 주요 알고리즘 : 회귀(Regression)와 분류(Classification)
    - 지도학습의 알고리즘은 크게 회귀(Regression)와 분류(Classification)로 나누어짐
      - 회귀
        - » 연속적인 숫자(값)를 예측하는 문제를
      - 분류
        - » 데이터를 미리 정의된 여러 범주(Category)로 나누는 문제를 해결

지도학습의 개념 및 지도학습 vs 비지도학습

- 머신러닝

  - 회귀 알고리즘(Regression)
    - 회귀는 연속적인 숫자 값을 예측하는 알고리즘
      - 예측 결과는 특정 숫자로 나타냄
      - 목표

        입력 데이터(Features)와 출력 값(Target) 간의 수학적 관계(선형/비선형 관계)를 모델링
    - 대표 알고리즘
      - **선형 회귀(Linear Regression)**

        → 입력 변수와 출력 변수 사이의 관계를 직선으로 표현

지도학습의 개념 및 지도학습 vs 비지도학습

- 머신러닝

  - 회귀 알고리즘(Regression)
    - 집값 예측
      - 문제: 집의 크기, 방 개수, 위치 등을 기반으로 집값을 예측.
      - 알고리즘: 선형 회귀를 사용해 입력 데이터와 집값 간의 관계를 학습.
      - 결과: 입력 데이터(Features)로 크기 40평, 방 3개, 도심 위치를 넣으면, 모델이 예상 집값(4억 원)을 예측.
    - 판매량 예측
      - 문제: 광고 비용을 바탕으로 제품 판매량을 예측.
      - 알고리즘: 선형 회귀를 사용해 "광고 비용과 판매량 간의 관계"를 학습.
      - 결과: 광고비를 100만 원 쓰면, 제품 판매량이 1000개일 것이라고 예측.

지도학습의 개념 및 지도학습 vs 비지도학습

- 머신러닝

  - 분류 알고리즘(Classification)
    - 분류는 데이터를 미리 정해진 범주(Category) 중 하나로 분류하는 알고리즘
      - 예측 결과는 숫자가 아닌 범주(Label)
      - 목표

        입력 데이터(Features)를 보고, 각 데이터가 어떤 범주에 속하는지 판단.
    - 대표 알고리즘
      - **선형 회귀(Linear Regression)**

        → 선형 회귀를 응용한 알고리즘으로, 출력 값을 특정 범주로 분류

        결과는 0~1 사이의 확률 값으로 나타나며,

        특정 기준(예: 0.5 이상)을 넘어가면 해당 범주로 분류.

지도학습의 개념 및 지도학습 vs 비지도학습

- 머신러닝

  - 분류 알고리즘(Classification)
    - 스팸 이메일 분류
      - 문제: 이메일의 내용을 기반으로 "스팸" 또는 "스팸 아님"으로 분류.
      - 알고리즘
        로지스틱 회귀를 사용해 이메일의 특징(특정 단어, 발신 도메인 등)과 라벨 간의 관계를 학습.
      - 결과: 이메일 내용에 "무료", "당첨" 같은 단어가 있으면 스팸으로 분류
    - 동물 분류
      - 문제: 사진을 보고, 해당 동물이 "강아지", "고양이", "새" 중 어느 범주에 속하는지 분류.
      - 알고리즘: 분류 알고리즘(예: 로지스틱 회귀)을 사용해 이미지 데이터와 범주 간의 관계를 학습.
      - 결과: 강아지의 특징(귀 모양, 코 모양 등)을 학습하고, 새로운 사진을 보고 강아지인지 판단.

지도학습의 개념 및 지도학습 vs 비지도학습

- 머신러닝

  - 회귀 vs 분류: 차이점

특징	회귀(Regression)	분류(Classification)
출력 형태	연속적인 숫자 값	미리 정의된 범주(Category)
목적	값 예측	데이터 분류
대표 알고리즘	선형 회귀, 다중 회귀	로지스틱 회귀, 서포트 벡터 머신(SVM), KNN
사례	집값 예측, 판매량 예측, 온도 예측	스팸 분류, 질병 진단, 이미지 분류

데이터의 역할 및 활용

- **데이터의 역할 및 활용**

  - **학습의 기반**
    - 모델이 학습하는 데 필요한 정보를 제공.
    - 데이터를 통해 입력(Features)과 출력(Target)의 관계를 파악.
  - **패턴 발견 및 예측 가능**
    - 데이터를 분석하여 숨겨진 패턴을 찾고, 이를 바탕으로 새로운 데이터를 예측.
    - 데이터가 없으면 패턴을 학습할 수도, 예측할 수도 없음.
  - **모델 평가**
    - 학습 후, 데이터를 통해 모델이 얼마나 정확한지 테스트.
    - 훈련 데이터와는 다른 검증 데이터를 사용하여 모델 성능을 평가.

first coding

데이터 준비 및 탐색

데이터 전처리 및 탐색적 데이터 분석(EDA)

학습 데이터와 테스트 데이터의 분리

pandas와 matplotlib를 활용한 데이터 시각화

데이터 전처리 및 탐색적 데이터 분석(EDA)

- **고객 만족도 설문 데이터 분석**

  - 상황
    - 한 회사가 고객 만족도 설문 데이터를 수집
    - 데이터는 다음과 같은 열(column)로 구성
      - CustomerID: 고객 ID
      - Age: 나이
      - Gender: 성별 (남성, 여성)
      - Satisfaction: 만족도 점수 (1~5, 1은 매우 불만족, 5는 매우 만족)
      - PurchaseAmount: 최근 구매 금액
    - 목표
      - 데이터를 분석하여 고객 만족도와 나이, 성별, 구매 금액의 관계를 이해하는 것

데이터 전처리 및 탐색적 데이터 분석(EDA)

- **고객 만족도 설문 데이터 분석**

  – 데이터 전처리

  (1) 결측치(Missing Values) 처리

  - 데이터를 살펴보니 Age와 PurchaseAmount 열에 일부 값이 누락되어 있음
  - 처리 방법
    - 평균값으로 채우기
      : Age의 결측치는 평균 나이로 대체.
    - 0으로 채우기
      : PurchaseAmount의 결측치는 0으로 대체.

```python
import pandas as pd

# 데이터 예시
data = {
    'CustomerID': [1, 2, 3, 4, 5],
    'Age': [25, 30, None, 35, 28],
    'Gender': ['Male', 'Female', 'Female', 'Male', None],
    'Satisfaction': [5, 4, 3, None, 2],
    'PurchaseAmount': [100, 200, None, 150, 0]
}
df = pd.DataFrame(data)
# 결측치 처리
# 나이는 평균으로 대체
df['Age'] = df['Age'].fillna(df['Age'].mean())
# 구매액은 0으로 대체
df['PurchaseAmount'] = df['PurchaseAmount'].fillna(0)
# 만족도 평균으로 대체
df['Satisfaction'] =
df['Satisfaction'].fillna(df['Satisfaction'].mean())

print(df)
```

	CustomerID	Age	Gender	Satisfaction	PurchaseAmount
0	1	25.0	Male	5.0	100.0
1	2	30.0	Female	4.0	200.0
2	3	29.5	Female	3.0	0.0
3	4	35.0	Male	3.5	150.0
4	5	28.0	None	2.0	0.0

데이터 전처리 및 탐색적 데이터 분석(EDA)

- **고객 만족도 설문 데이터 분석**

  – 데이터 전처리

  (2) 이상치(Outliers) 처리

  - 구매 금액이 너무 높은 경우(예: 1억 원), 이상치로 간주할 수 있음
  - 처리 방법

    : 특정 기준(예: IQR)을 사용해 이상치를 제거하거나 조정

```python
import pandas as pd

# 데이터 예시
data = {
    'CustomerID': [1, 2, 3, 4, 5],
    'Age': [25, 30, None, 35, 28],
    'Gender': ['Male', 'Female', 'Female', 'Male', None],
    'Satisfaction': [5, 4, 3, None, 2],
    'PurchaseAmount': [100, 200, None, 150, 0]
}
df = pd.DataFrame(data)

# 결측치 처리
df['Age'].fillna(df['Age'].mean(), inplace=True)
df['PurchaseAmount'].fillna(0, inplace=True)
print(df)
```

```
   CustomerID   Age  Gender  Satisfaction  PurchaseAmount
0           1  25.0    Male           5.0           100.0
1           2  30.0  Female           4.0           200.0
2           3  29.5  Female           3.0             0.0
3           4  35.0    Male           NaN           150.0
4           5  28.0    None           2.0             0.0
```

데이터 전처리 및 탐색적 데이터 분석(EDA)

- **고객 만족도 설문 데이터 분석**

  – 탐색적 데이터 분석 (EDA)

  (1) 데이터의 기초 통계 파악

  - 평균, 중앙값, 최댓값, 최솟값을 확인해 데이터를 요약

```
…

df = pd.DataFrame(data)

# 결측치 처리
df['Age'].fillna(df['Age'].mean(), inplace=True)
df['PurchaseAmount'].fillna(0, inplace=True)
# print(df)

# 데이터 요약
print(df.describe())
```

	CustomerID	Age	Satisfaction	PurchaseAmount
count	5.000000	5.000000	5.000000	5.000000
mean	3.000000	29.500000	3.500000	90.000000
std	1.581139	3.640055	1.118034	89.442719
min	1.000000	25.000000	2.000000	0.000000
25%	2.000000	28.000000	3.000000	0.000000
50%	3.000000	29.500000	3.500000	100.000000
75%	4.000000	30.000000	4.000000	150.000000
max	5.000000	35.000000	5.000000	200.000000

데이터 전처리 및 탐색적 데이터 분석(EDA)

- **고객 만족도 설문 데이터 분석**

  – 탐색적 데이터 분석 (EDA)

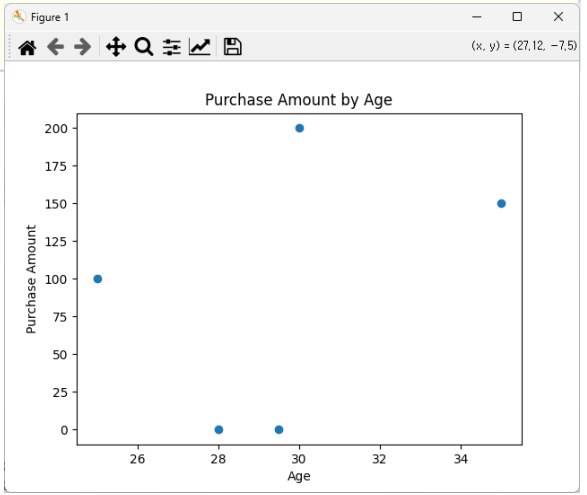
  (2) 데이터 시각화로 이해

  - 나이에 따른 구매 금액 분포

    나이와 구매 금액 간의 관계를 산점도로 표현

```python
import matplotlib.pyplot as plt

# 나이 vs 구매 금액 산점도
plt.scatter(df['Age'], df['PurchaseAmount'])
plt.title('Purchase Amount by Age')
plt.xlabel('Age')
plt.ylabel('Purchase Amount')
plt.show()
```



데이터 전처리 및 탐색적 데이터 분석(EDA)

- **고객 만족도 설문 데이터 분석**

  – 탐색적 데이터 분석 (EDA)

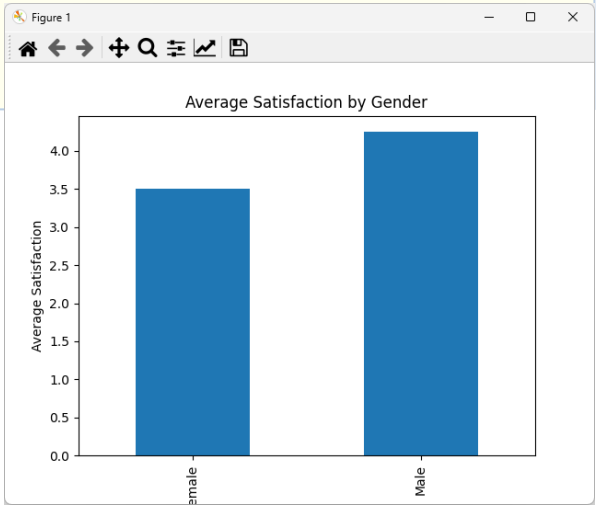
  (2) 데이터 시각화로 이해

  - 성별에 따른 만족도

    남성과 여성의 만족도 평균을 비교

```python
import matplotlib.pyplot as plt

# 성별별 만족도 평균 시각화
gender_satisfaction =
df.groupby('Gender')['Satisfaction'].mean()
gender_satisfaction.plot(kind='bar')
plt.title('Average Satisfaction by Gender')
plt.xlabel('Gender')
plt.ylabel('Average Satisfaction')
plt.show()
```



학습 데이터와 테스트 데이터의 분리

- **학습 데이터와 테스트 데이터의 분리**

  - 머신러닝 모델이 데이터를 학습(Training)하는 과정과 평가(Test)하는 과정을 분리하기 위해 필요
    - 학습 데이터

      모델이 학습하는 데 사용하는 데이터.
    - 테스트 데이터

      학습이 끝난 후 모델의 성능을 평가하는 데 사용하는 데이터.

학습 데이터와 테스트 데이터의 분리

- **학습 데이터와 테스트 데이터의 분리**

  - 왜 학습 데이터와 테스트 데이터를 분리해야 할까?

    - **모델 과적합(Overfitting) 방지**
      - 모델이 학습 데이터에 너무 특화되면, 새로운 데이터(테스트 데이터)에 대해 좋은 성능을 내지 못할 수 있음
      - 학습 데이터와 테스트 데이터를 분리하면 이런 문제를 예방할 수 있음
    - **모델의 일반화 능력 평가**
      - 모델이 학습하지 않은 데이터에 대해 얼마나 잘 예측할 수 있는지를 평가할 수 있음

학습 데이터와 테스트 데이터의 분리

- 데이터 분리 과정
  - 특성과 타겟 분리
    - **특성(Feature)**
      예측에 사용할 데이터
      (Age, Gender, PurchaseAmount)
    - **타겟(Target)**
      예측하고자 하는 값
      (Satisfaction)

	CustomerID	Age	Gender	Satisfaction	PurchaseAmount
0	1	25.0	Male	5.0	100.0
1	2	30.0	Female	4.0	200.0
2	3	29.5	Female	3.0	0.0
3	4	35.0	Male	NaN	150.0
4	5	28.0	None	2.0	0.0

```python
# Gender는 숫자로 변환 (Label Encoding)
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})

# 특성과 타겟 분리
X = df[['Age', 'Gender', 'PurchaseAmount']]  # 입력 데이터
y = df['Satisfaction']  # 출력 데이터

print(X)
print(y)
```

```
    Age  Gender  PurchaseAmount
0  25.0     0.0           100.0
1  30.0     1.0           200.0
2  29.5     1.0             0.0
3  35.0     0.0           150.0
4  28.0     NaN             0.0
0    5.0
1    4.0
2    3.0
3    3.5
4    2.0
```

학습 데이터와 테스트 데이터의 분리

- 데이터 분리 과정

  – 학습 데이터와 테스트 데이터 분리

    - 데이터셋을
      학습용(80%)과
      테스트용(20%)으로 분리
    - train_test_split 함수 사용.

```python
from sklearn.model_selection import train_test_split

# 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print("학습 데이터 크기:", X_train.shape)
print("테스트 데이터 크기:", X_test.shape)

print(X_train) # 학습 데이터의 입력 데이터
print('*'*10)
print(y_train) # 학습 데이터의 출력 데이터
print('*'*10)
print(X_test)  # 테스트 데이터의 입력 데이터
print('*'*10)
print(y_test)  # 테스트 데이터의 출력 데이터
```

```
학습 데이터 크기: (4, 3)
테스트 데이터 크기: (1, 3)
    Age  Gender  PurchaseAmount
4  28.0     NaN             0.0
2  29.5     1.0             0.0
0  25.0     0.0           100.0
3  35.0     0.0           150.0
**********
4    2.0
2    3.0
0    5.0
3    3.5
Name: Satisfaction, dtype: float64
```

```
**********
    Age  Gender  PurchaseAmount
1  30.0     1.0           200.0
**********
1    4.0
Name: Satisfaction, dtype: float64
```

first coding

지도학습 - 회귀 알고리즘

회귀 알고리즘의 이해

선형 회귀의 원리 및 모델 학습

손실 함수(MSE)와 성능 평가(Metrics)

회귀 알고리즘의 이해

- **회귀 알고리즘의 이해**

  - 회귀(Regression)
    연속적인 숫자 데이터를 예측하는 데 사용되는 지도학습(Supervised Learning) 알고리즘
    → 입력 데이터를 기반으로 숫자 값을 예측하는 문제를 해결하는 방법

  - 예시
    - 집값 예측
      - 집의 면적, 방 개수, 위치 등의 정보를 입력하면 집의 가격을 예측.
    - 날씨 예측
      - 시간, 온도, 습도 데이터를 기반으로 다음날의 기온을 예측.
    - 회귀 알고리즘은 입력 데이터와 출력 값 사이의 관계를 모델링하여 새로운 데이터를 기반으로 값을 예측

회귀 알고리즘의 이해

- **회귀 알고리즘의 이해**

  - 회귀(Regression)

    연속적인 숫자 데이터를 예측하는 데 사용되는 지도학습(Supervised Learning) 알고리즘

    → 입력 데이터를 기반으로 숫자 값을 예측하는 문제를 해결하는 방법

  - 회귀의 특징

    - 연속적인 출력 값

      예측 값은 숫자이며, 범주형 데이터(예: 고양이, 개)가 아님

    - 독립 변수와 종속 변수의 관계

      독립 변수(입력 데이터)가 종속 변수(예측 값)에 어떤 영향을 주는지 분석

회귀 알고리즘의 이해

- **회귀 알고리즘의 이해**

  - 회귀의 종류
    - 단순 선형 회귀
      - 입력 변수가 하나인 경우.

        예: 공부 시간에 따른 시험 점수 예측.그래프 상에서 데이터는 직선 형태로 표현
    - 다중 선형 회귀
      - 입력 변수가 여러 개인 경우.

        예: 집값 예측 (집 크기, 위치, 층수 등을 입력).데이터 관계가 더 복잡해질 수 있음
    - 비선형 회귀
      - 입력 데이터와 출력 값 사이의 관계가 선형이 아닌 경우.

        예: 바이러스 확산 모델, 곡선 형태의 데이터.

- **회귀 알고리즘의 이해**

  - 회귀는 어떻게 동작하나?

    ① 학습 데이터 수집

    모델이 배울 수 있도록 입력 데이터(독립 변수)와 정답(종속 변수)을 준비

    ② 모델 학습

    데이터를 이용해 입력과 출력 사이의 수학적 관계를 찾음

    ③ 예측

    학습한 모델을 사용해 새로운 데이터를 입력하면 결과 값을 예측

선형 회귀의 원리 및 모델 학습

- 선형 회귀는 데이터 간의 직선 관계를 찾는 알고리즘
- 입력 데이터(독립 변수)가 증가하거나 감소할 때 출력 값(종속 변수)이 어떻게 변하는지 예측하는 데 사용

- 사례

  공부 시간과 시험 점수 예측

  학생들이 공부한 시간(입력)을 가지고

  시험 점수(출력)를 예측하고 싶다!

공부 시간 (X)	시험 점수 (Y)
1시간	50점
2시간	55점
3시간	65점
4시간	70점
5시간	75점

```python
import numpy as np
import matplotlib.pyplot as plt

# 입력 데이터 (공부 시간)
X = np.array([1, 2, 3, 4, 5])

# 출력 데이터 (시험 점수)
y = np.array([50, 55, 65, 70, 75])
```

선형 회귀의 원리 및 모델 학습

- **선형 회귀의 원리 기본 원리**

  – 선형 회귀는 데이터를

  y = mx + b라는 직선 방정식으로 표현

  - x: 독립 변수 (공부 시간)
  - y: 종속 변수 (시험 점수)
  - m: 기울기 (공부 시간이 늘어나면 점수가 얼마나 증가하는지 나타냄)
  - b: y절편 (공부 시간이 0일 때의 점수)

  – 목표

  데이터를 가장 잘 설명할 수 있는

  m(기울기)와 b(y절편)를 찾아 직선을 만듦

```
# 2. 초기 값 설정
m = 0  # 초기 기울기
b = 0  # 초기 y절편
learning_rate = 0.01  # 학습 속도
epochs = 1000  # 학습 반복 횟수

# 경사하강법을 이용한 학습
for _ in range(epochs):
    # 예측 값 계산
    y_pred = m * X + b

    # 손실 함수(MSE) 계산
    error = y - y_pred
    mse = (error ** 2).mean()

    # 기울기(m)와 y절편(b)의 변화량 계산
    m_gradient = -(2 / len(X)) * sum(X * error)
    b_gradient = -(2 / len(X)) * sum(error)

    # 기울기와 절편 업데이트
    m -= learning_rate * m_gradient
    b -= learning_rate * b_gradient
```

선형 회귀의 원리 및 모델 학습

- **모델 학습 과정**

1) 데이터 시각화

   - 먼저 데이터를 그래프에 표시.
     각 점은 학생 한 명의 공부 시간과 시험 점수
     - x축: 공부 시간, y축: 시험 점수
     - 이 데이터 위에 최적의 직선을 그려야 함

2) 최적의 직선 찾기

   - 직선을 그릴 때 중요한 점
     - 직선이 데이터 점과 얼마나 가까운지를 측정
       → 이 거리(오차)를 측정하는 방법이 **손실 함수**
     - 점에서 직선까지의 거리를 최소화하는 직선을 찾아야 함!

선형 회귀의 원리 및 모델 학습

- **모델 학습 과정**

3) 손실 함수: 평균 제곱 오차(MSE)

MSE (Mean Squared Error)는 데이터 점과 직선 사이의 거리(오차)의 제곱 평균을 계산

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\hat{y}_i))^2$$

- yi : 실제 점수 (데이터 값)
- yi ^ : 모델이 예측한 점수 (직선 값)
- n : 데이터의 개수

- 직선 y=5x+45 을 사용해
  예측한 점수와 실제 점수의 차이를 계산
  - **공부 시간 1시간** → 예측 점수: 5(1)+45=50
    (실제: 50, 오차: 0)
  - **공부 시간 2시간** → 예측 점수: 5(2)+45=55
    (실제: 55, 오차: 0)
- 모든 데이터에 대해 계산한 오차를 평균 내어 MSE를 구함

선형 회귀의 원리 및 모델 학습

- **모델 학습 과정**

4) 기울기와 y절편 업데이트

- 경사하강법(Gradient Descent)
  MSE를 최소화하는 방향으로 기울기(m)와 y절편(b)을 조금씩 조정
- 반복적으로 업데이트하면서 오차가 최소화될 때까지 학습

5) 모델 학습의 결과

최종적으로 $y=mx+b$y에 대해

- m=5 (기울기): 공부 시간이 1시간 늘면 시험 점수가 평균 5점 증가
- b=45 (y절편): 공부를 전혀 하지 않아도 시험 점수는 45점

선형 회귀의 원리 및 모델 학습

- **모델 학습 과정**

6) 새로운 데이터 예측

- 학습된 모델로 새로운 데이터를 예측할 수 있음
  - 공부 시간 6시간 → 예측 점수: y=5(6)+45=75
  - 공부 시간 8시간 → 예측 점수: y=5(8)+45=85

선형 회귀의 원리 및 모델 학습

- 모델 학습 과정

```python
import numpy as np
import matplotlib.pyplot as plt

# 1. 데이터 준비
X = np.array([1, 2, 3, 4, 5])  # 공부 시간 (입력 데이터)
y = np.array([50, 55, 65, 70, 75])  # 시험 점수 (출력 데이터)

# 2. 초기 값 설정
m = 0  # 초기 기울기
b = 0  # 초기 y절편
learning_rate = 0.01  # 학습 속도
epochs = 1000  # 학습 반복 횟수

# 3. 경사하강법을 이용한 학습
for _ in range(epochs):
    # 예측 값 계산
    y_pred = m * X + b

    # 손실 함수(MSE) 계산
    error = y - y_pred
    mse = (error ** 2).mean()

    # 기울기(m)와 y절편(b)의 변화량 계산
    m_gradient = -(2 / len(X)) * sum(X * error)
    b_gradient = -(2 / len(X)) * sum(error)

    # 기울기와 절편 업데이트
    m -= learning_rate * m_gradient
    b -= learning_rate * b_gradient
```

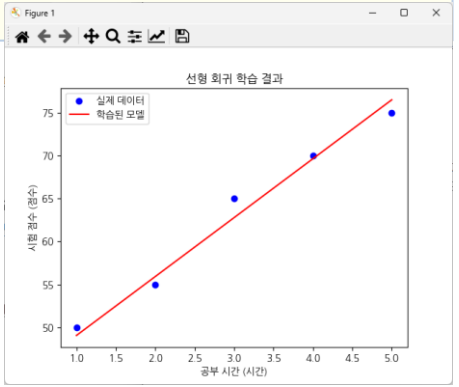
```python
# 학습 결과 출력
print(f"학습된 기울기(m): {m:.2f}")
print(f"학습된 y절편(b): {b:.2f}")

# 4. 학습 결과 시각화
y_pred = m * X + b  # 학습된 모델의 예측 값

plt.rc('font', family='NanumGothic') # For Windows

plt.scatter(X, y, color='blue', label='실제 데이터')  # 실제 데이터
plt.plot(X, y_pred, color='red', label='학습된 모델')  # 학습된 직선
plt.xlabel('공부 시간 (시간)')
plt.ylabel('시험 점수 (점수)')
plt.legend()
plt.title('선형 회귀 학습 결과')
plt.show()

# 5. 새로운 데이터 예측
new_study_hours = np.array([6, 8, 10])  # 새로운 공부 시간
predicted_scores = m * new_study_hours + b
print("새로운 공부 시간에 따른 예측 점수:", predicted_scores)
```



- 성능 평가(Metrics)

  - 실제 점수 (y)와 모델이 예측한 점수 (y_pred)를 비교

공부 시간 (X)	실제 점수 (Y)	예측 점수 $y^$	오차(실제 - 예측)
1시간	50점	50	0
2시간	55점	54	1
3시간	65점	65	0
4시간	70점	71	-1
5시간	75점	75	0

성능 평가(Metrics)

- 주요 성능 평가 지표
  - MSE (Mean Squared Error)
    - 평균 제곱 오차MSE는 오차(예측 값과 실제 값의 차이)의 제곱 평균
    - 오차가 크면 제곱이 적용되므로 더 큰 패널티를 줌
    - MSE 값이 작을수록 모델이 더 정확함

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- y_i: 실제 값
- $\hat{y}_i$: 예측 값
- n: 데이터 개수

$$MSE = \frac{1}{5}[(50-50)^2 + (55-54)^2 + (65-65)^2 + (70-71)^2 + (75-75)^2] = \frac{1}{5}[0+1+0+1+0] = 0.4$$

성능 평가(Metrics)

- 주요 성능 평가 지표
  - RMSE (Root Mean Squared Error)
    - 평균 제곱근 오차MSE의 제곱근을 계산한 값
    - 단위를 실제 값과 동일하게 만들어 해석이 상대적으로 쉬움

$$RMSE = \sqrt{MSE}$$

$$RMSE = \sqrt{0.4} = 0.63$$

    - 예측 값과 실제 값의 평균적인 오차는 약 0.63

성능 평가(Metrics)

- 주요 성능 평가 지표
  - MAE (Mean Absolute Error): 평균 절대 오차
    - 오차의 절대값 평균을 계산
    - 제곱을 사용하지 않으므로 큰 오차의 영향이 덜함

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$MAE = \frac{1}{5}[|50-50| + |55-54| + |65-65| + |70-71| + |75-75|] = \frac{1}{5}[0+1+0+1+0] = 0.4$$

    - 평균적으로 모델은 실제 값과 0.4점 정도 차이

성능 평가(Metrics)

- 주요 성능 평가 지표
  - R^2 (R-squared) : 결정 계수
    - R^2 은 모델이 데이터를 얼마나 잘 설명하는지를 나타냄
    - 0에서 1 사이의 값을 가지며, 1에 가까울수록 모델이 데이터를 잘 설명함

$$R^2 = 1 - \frac{\text{SS}_{res}}{\text{SS}_{tot}}$$

SSres : 잔차 제곱합 (예측 값과 실제 값의 차이)

SStot : 총 제곱합 (실제 값과 평균 값의 차이)

$$\text{SS}_{res} = \sum (y_i - \hat{y}_i)^2 = 0 + 1 + 0 + 1 + 0 = 2$$

$$\text{SS}_{tot} = \sum (y_i - \bar{y})^2 = (50 - 63)^2 + (55 - 63)^2 + (65 - 63)^2 + (70 - 63)^2 + (75 - 63)^2 = 178$$

$$R^2 = 1 - \frac{\text{SS}_{res}}{\text{SS}_{tot}} = 1 - \frac{2}{178} \approx 0.99$$

- 모델은 데이터를 약 99% 설명

성능 평가(Metrics)

- **주요 성능 평가 지표**

지표	값	해석
MSE	0.4	평균적으로 제곱된 오차가 작습니다.
RMSE	0.63	모델의 평균 오차는 약 0.63점입니다.
MAE	0.4	모델은 평균적으로 약 0.4점 차이로 예측합니다.
R^2	0.99	모델이 데이터를 매우 잘 설명합니다.

- MSE : 큰 오차에 민감한 평가 방법 (작은 실수를 크게 보려는 돋보기 같은 역할)
- MAE : 절대적인 평균 오차를 단순히 계산 (실수의 평균 크기를 직접 보는 역할)
- RMSE : MSE의 단위를 실제 값과 맞추어 해석하기 쉽게 만든 것
- R^2 : 모델이 데이터를 설명하는 정도를 백분율로 나타낸 점

성능 평가(Metrics)

- Scikit-learn을 사용한 선형 회귀 학습

① 데이터 준비

- X는 공부 시간을 입력으로 사용하며 2D 배열로 변환(reshape(-1, 1))하여 학습

② 모델 학습

- LinearRegression 객체를 생성한 후 fit 메서드를 사용해 학습

③ 성능 평가

- mean_squared_error: 평균 제곱 오차(MSE)를 계산
- mean_absolute_error: 평균 절대 오차(MAE)를 계산
- r2_score: 결정 계수($R2$R 2)를 계산

④ 결과 시각화

- 실제 데이터는 파란 점으로 표시하고, 학습된 직선은 빨간 선으로 시각화합니다.새로운 데이터

⑤ 예측

성능 평가(Metrics)

- Scikit-learn을 사용한 선형 회귀 학습

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# 1. 데이터 준비
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)  # 공부 시간 (2D 배열로 변환)
y = np.array([50, 55, 65, 70, 75])  # 시험 점수

# 2. 모델 학습
model = LinearRegression()  # 선형 회귀 모델 생성
model.fit(X, y)  # 모델 학습

# 3. 학습된 모델의 기울기와 절편 확인
m = model.coef_[0]  # 기울기
b = model.intercept_  # 절편
print(f"학습된 기울기(m): {m:.2f}")
print(f"학습된 y절편(b): {b:.2f}")

# 4. 예측 값 계산
y_pred = model.predict(X)  # 학습 데이터에 대한 예측 값

# 5. 성능 평가
mse = mean_squared_error(y, y_pred)  # MSE 계산
rmse = np.sqrt(mse)  # RMSE 계산
mae = mean_absolute_error(y, y_pred)  # MAE 계산
r2 = r2_score(y, y_pred)  # R^2 계산

print(f"MSE: {mse:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"MAE: {mae:.2f}")
print(f"R^2: {r2:.2f}")
```

```python
# 6. 학습 결과 시각화
plt.rc('font', family='NanumGothic') # For Windows
plt.scatter(X, y, color='blue', label='실제 데이터')  # 실제 데이터
plt.plot(X, y_pred, color='red', label='학습된 모델')  # 학습된 직선
plt.xlabel('공부 시간 (시간)')
plt.ylabel('시험 점수 (점수)')
plt.legend()
plt.title('Scikit-learn을 이용한 선형 회귀 학습 결과')
plt.show()

# 7. 새로운 데이터 예측
new_study_hours = np.array([6, 8, 10]).reshape(-1, 1)  # 새로운 공부 시간
predicted_scores = model.predict(new_study_hours)  # 예측
print("새로운 공부 시간에 따른 예측 점수:", predicted_scores)
```

```
학습된 기울기(m): 6.50
학습된 y절편(b): 43.50
MSE: 1.50
RMSE: 1.22
MAE: 1.00
R^2: 0.98
새로운 공부 시간에 따른 예측 점수: [ 82.5  95.5 108.5]
```

성능 평가(Metrics)

- **과적합(Overfitting)과 과소적합(Underfitting)**

  - 모델 학습 과정에서 성능과 관련된 문제
  - 모델이 데이터에 너무 치우치거나 일반화를 잘하지 못할 때 발생
  - 사례
    - 시험 점수 예측 모델데이터

      학생들이 공부한 시간(X)에 따라 시험 점수(y)를 예측하는 모델을 만들고 있다고 가정

공부 시간 (X)	시험 점수 (Y)
1시간	50점
2시간	55점
3시간	65점
4시간	70점
5시간	75점

성능 평가(Metrics)

- **과소적합(Underfitting)**
  - 데이터를 제대로 학습하지 못한 상태
    - 모델이 데이터의 패턴을 충분히 학습하지 못함.
    - 학습 데이터와 테스트 데이터 모두에서 성능이 나쁨.
    - 단순한 모델로 인해 데이터의 복잡한 관계를 놓침.
  - 사례
    - 선형 회귀 모델이 y = 60 같은 단순한 직선을 학습했다고 가정.
    - 모든 입력 데이터에 대해 동일한 점수(60점)를 예측. → 모델이 데이터의 변화를 반영하지 못해 성능이 낮음

공부 시간 (X)	실제 점수 (Y)	예측 점수 $y^$	오차(실제 - 예측)
1시간	50점	60	-10
2시간	55점	60	-5
3시간	65점	60	5
4시간	70점	60	10
5시간	75점	60	15

성능 평가(Metrics)

- 과적합(Overfitting)
  - 데이터를 너무 많이 학습한 상태
    - 모델이 학습 데이터에 너무 집착해 데이터의 작은 노이즈까지 학습.
    - 학습 데이터에서는 성능이 좋지만, 새로운 데이터(테스트 데이터)에서는 성능이 나쁨.
    - 복잡한 모델로 인해 일반화 능력이 부족.
  - 사례
    - 모델이 학습 데이터를 완벽히 맞추기 위해 다항식(Polynomial) 같은 복잡한 관계를 학습.
    - 학습 데이터는 잘 예측하지만, 새로운 데이터에는 잘 맞지 않음.

공부 시간 (X)	실제 점수 (Y)	예측 점수 y^	오차(실제 - 예측)
1시간	50점	50	0
2시간	55점	55	0
3시간	65점	65	0
4시간	70점	70	0
5시간	75점	75	0

성능 평가(Metrics)

- 과적합(Overfitting)과 과소적합(Underfitting)

항목	과소적합(Underfitting)	과적합(Overfitting)
학습 데이터 성능	낮음	매우 높음
테스트 데이터 성능	낮음	낮음
모델 복잡성	너무 단순	너무 복잡
일반화 능력	부족	부족

성능 평가(Metrics)

- **과적합(Overfitting)과 과소적합(Underfitting)**

  - 해결 방법
    - **과소적합 해결 방법**
      - 모델을 더 복잡하게 만듦 (더 많은 특징 추가, 더 큰 모델 사용)
      - 학습 데이터를 더 많이 수집.학습 시간을 늘려 모델을 충분히 학습
    - **과적합 해결 방법**
      - 모델의 복잡도를 낮춤 (간단한 모델 사용, 차원 축소)
      - 규제(Regularization) 적용
        - » L1 규제: 불필요한 특징 제거
        - » L2 규제: 가중치를 너무 크게 만드는 것을 방지
      - 더 많은 데이터를 수집하거나, 교차 검증(Cross Validation) 사용

first coding

지도학습 - 분류 알고리즘

분류 문제의 개념

로지스틱 회귀(Logistic Regression) 이해

성능 평가 지표(정확도, Precision, Recall, F1-Score)

분류 문제의 개념

- **분류 문제**

  - **분류(Classification)**
    - 주어진 데이터를 특정 범주(Category)나 클래스(Class)로 분류하는 작업
    - 지도 학습(Supervised Learning)의 한 유형 → 데이터에 레이블(정답)이 존재
    - 모델은 이 레이블을 학습하여 새로운 데이터가 주어졌을 때 올바른 클래스를 예측하도록 훈련됨

  - **분류의 주요 목표**
    - 정확한 예측 : 데이터가 어떤 클래스에 속하는지 최대한 정확히 예측.
    - 실용적 적용 : 실제 문제에서 유용한 정보를 제공.
      예: 스팸 필터링 모델은 높은 정확도와 낮은 오탐(False Positive)을 목표로 함.

분류 문제의 개념

- **분류 문제**

  - 분류 문제의 특징
    - **입력 데이터** - 다양한 특징(feature)을 가진 데이터. (예: 이메일 본문의 단어들)
    - **출력 데이터** - 데이터가 속한 범주 또는 클래스. (예: 이메일이 "스팸"인지 "스팸 아님"인지)
    - **클래스 개수**
      - **이진 분류(Binary Classification)**
        : 두 개의 클래스만 예측 (예: True/False, 0/1).
      - **다중 클래스 분류(Multi-Class Classification)**
        : 세 개 이상의 클래스 예측 (예: 고양이, 강아지, 새).

분류 문제의 개념

- **분류의 작동 방식**

① **데이터 수집 및 전처리**

- 데이터를 수집하고, 결측값 처리, 스케일링, 정규화 등을 통해 준비.

② **모델 학습**

- 입력 데이터(특징, Features)와 레이블(정답, Labels)을 활용해 모델을 학습시킴.

  예: 키와 몸무게를 입력으로 받아 성별을 예측하는 모델.

③ **모델 예측**

- 새로운 데이터가 들어오면 학습한 모델이 예측 값을 출력.

  예: "이 메일은 스팸이다" 또는 "이 환자는 암이 아니다."

④ **평가**

- 모델의 예측 결과를 평가하여 성능 지표(정확도, Precision, Recall 등)로 측정.

로지스틱 회귀(Logistic Regression) 이해

- **로지스틱 회귀(Logistic Regression)**

  - 분류 문제를 해결하기 위한 지도 학습 알고리즘

  - 이름에 "회귀"가 포함되어 있지만,
    실제로는 연속적인 값을 예측하는 것이 아니라 데이터를 클래스로 분류하는 데 사용

  - 로지스틱 회귀는 이진 분류 문제에서 가장 널리 사용되는 기법 중 하나

  - 데이터를 분석하고 특정 데이터가 특정 클래스에 속할 확률을 예측
    예) 로지스틱 회귀는 "이 이메일이 스팸일 확률이 85%"와 같은 값을 출력하며,
    이 확률에 따라 특정 클래스에 할당

로지스틱 회귀(Logistic Regression) 이해

- **로지스틱 회귀(Logistic Regression) 의 작동 원리**

① 선형 회귀의 확장

- 로지스틱 회귀는 먼저 선형 회귀처럼 데이터를 학습
- 선형 회귀의 예측 값(연속 값)을 분류 문제에 적합하도록 변환
- 문제: 선형 회귀는 0에서 1 사이의 확률 값을 보장하지 않음.

② 시그모이드 함수 적용

- 로지스틱 회귀는 예측 값 z를 **시그모이드 함수를 사용해 0과 1 사이의 값(확률)으로 변환**
- 시그모이드 함수

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$z=w1x1+w2x2+\cdots+wnxn+b$: 입력 특징들의 가중함.

시그모이드 함수의 출력 값은 항상 $0 \le \sigma(z) \le 1$

③ 클래스 결정

- 시그모이드 함수의 결과(확률 값)가 0.5 이상이면 1, 그렇지 않으면 0으로 분류 → 임계값 변경 가능

로지스틱 회귀(Logistic Regression) 이해

- **로지스틱 회귀(Logistic Regression) 의 과정**

① 가설 설정

- 입력 데이터의 특징을 선형 결합하여 z를 계산하고, 이를 시그모이드 함수에 통과시켜 확률을 계산
- 모델 출력 $h\theta(x)=\sigma(z)$

② 손실 함수 정의

- 로지스틱 회귀는 로그 손실 함수(Log Loss)를 사용

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\theta(x_i)) + (1-y_i)\log(1-h_\theta(x_i))]$$

여기서 yiy i 는 실제 값, $h\theta(xi)$h θ (x i)는 예측 값

목표: 손실 함수 값을 최소화하여 모델 성능 최적화.

③ 모델 학습

- 경사 하강법(Gradient Descent)을 사용하여 가중치와 절편 값을 업데이트.
- 모델이 점점 더 정확하게 데이터를 분류하도록 학습.

성능 평가 지표(정확도, Precision, Recall, F1-Score)

- **성능 평가 지표**

  - 분류 모델의 성능을 평가하기 위해 다양한 평가 지표가 사용
  - 각각의 지표는 모델의 성능을 다양한 관점에서 분석하며, 특정 문제에 더 적합한 지표를 선택 해야함

  - 혼동 행렬 (Confusion Matrix)
    - 모든 평가 지표는 혼동 행렬(Confusion Matrix)을 기반으로 계산
  - 지표 종류
    - 정확도(Accuracy)
    - 정밀도(Precision)
    - 재현율(Recall)
    - F1-Score

성능 평가 지표(정확도, Precision, Recall, F1-Score)

- **성능 평가 지표**
  - 혼동 행렬 (Confusion Matrix)

항목	실제 Positive	실제 Negative
예측 Positive	True Positive (TP)	False Positive (FP)
예측 Negative	False Negative (FN)	True Negative (TN)

    - 문제 상황
      - Positive: 암 환자 (실제 암 환자)
      - Negative: 암이 아닌 환자 (실제 암이 아닌 환자)
      - 모델은 각 환자에 대해 암 여부를 "양성(Positive)" 또는 **"음성(Negative)"**으로 예측
    - 혼동 행렬 구성 요소
      - True Positive (TP) : 모델이 암 환자를 암이라고 정확히 예측한 경우
      - True Negative (TN) : 모델이 암이 아닌 환자를 암이 아니라고 정확히 예측한 경우
      - False Positive (FP) : (Type I Error)모델이 암이 아닌 환자를 암이라고 잘못 예측한 경우
      - False Negative (FN): (Type II Error)모델이 암 환자를 암이 아니라고 잘못 예측한 경우

성능 평가 지표(정확도, Precision, Recall, F1-Score)

- **성능 평가 지표**

  - 혼동 행렬 (Confusion Matrix)

    - 모델이 100명의 환자 데이터를 예측한 결과

      - 실제 암 환자 (**Positive**): 40명
      - 실제 암이 아닌 사람 (**Negative**): 60명
      - 모델의 예측 결과

        » TP = 30명 (암 환자 40명 중 30명을 정확히 예측)

        » TN = 50명 (암이 아닌 사람 60명 중 50명을 정확히 예측)

        » FP = 10명 (암이 아닌 사람 60명 중 10명을 암으로 잘못 예측)

        » FN = 10명 (암 환자 40명 중 10명을 암이 아니라고 잘못 예측)

항목	실제 Positive	실제 Negative
예측 Positive	30 (TP)	10 (FP)
예측 Negative	10 (FN)	50 (TN)

성능 평가 지표(정확도, Precision, Recall, F1-Score)

- 성능 평가 지표

항목	실제 Positive	실제 Negative
예측 Positive	30 (TP)	10 (FP)
예측 Negative	10 (FN)	50 (TN)

  - 정확도 (Accuracy)
    - 정확도는 모델이 올바르게 예측한 비율을 의미

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad \frac{30 + 50}{30 + 50 + 10 + 10} = 0.8(80\%)$$

    - 장점 : 전체적인 예측 성능을 간단히 측정할 수 있음.
    - 단점: 데이터의 클래스 불균형(Positive와 Negative의 비율이 크게 차이 날 때)에서 신뢰도가 낮아질 수 있음.
  - 예시

    암 진단에서 대부분의 환자가 건강(99%)이라면,

    모델이 항상 "건강"이라고 예측해도 99% 정확도를 얻을 수 있지만 이는 의미 있는 성능이 아님.

성능 평가 지표(정확도, Precision, Recall, F1-Score)

- **성능 평가 지표**

항목	실제 Positive	실제 Negative
예측 Positive	30 (TP)	10 (FP)
예측 Negative	10 (FN)	50 (TN)

  - 정밀도 (Precision)
    - 정밀도는 모델이 Positive로 예측한 것들 중에서 실제로 Positive인 비율

$$Precision = \frac{TP}{TP + FP} \qquad \frac{30}{30 + 10} = 0.75(75\%)$$

    - 장점 : False Positive를 줄이는 데 중점을 둠.
      예: 스팸 필터링에서 정밀도가 높을수록, 스팸이 아닌 이메일을 스팸으로 잘못 분류할 가능성이 낮아짐.
    - 단점 : Positive 데이터를 놓칠 수 있음 (Recall이 낮아질 수 있음).

성능 평가 지표(정확도, Precision, Recall, F1-Score)

- 성능 평가 지표

항목	실제 Positive	실제 Negative
예측 Positive	30 (TP)	10 (FP)
예측 Negative	10 (FN)	50 (TN)

  - 재현율 (Recall, Sensitivity, TPR)
    - 재현율은 실제 Positive 중에서 모델이 Positive로 올바르게 예측한 비율

$$Recall = \frac{TP}{TP + FN} \qquad \frac{30}{30 + 10} = 0.75(75\%)$$

    - 장점 : False Negative를 줄이는 데 중점을 둠.
      예: 암 진단 모델에서 재현율이 높으면, 암 환자를 놓칠 확률이 낮아짐.
    - 단점 : Positive로 예측하는 경향이 커져 정밀도가 낮아질 수 있음.

성능 평가 지표(정확도, Precision, Recall, F1-Score)

- 성능 평가 지표

항목	실제 Positive	실제 Negative
예측 Positive	30 (TP)	10 (FP)
예측 Negative	10 (FN)	50 (TN)

  - F1-Score
    - 정밀도(Precision)와 재현율(Recall)의 조화 평균(Harmonic Mean)으로, 두 지표의 균형을 평가

$$F1\text{-}Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \qquad 2 \times \frac{0.75 \times 0.75}{0.75 + 0.75} = 0.75(75\%)$$

    - 장점 : 정밀도와 재현율 중 어느 하나에 치우치지 않고, 종합적인 성능 평가 가능.
    - 단점 : 클래스 불균형이 심한 데이터에서는 추가적인 분석이 필요.

성능 평가 지표(정확도, Precision, Recall, F1-Score)

- **성능 평가 지표**

  - 지표 선택 기준
    - 정확도(Accuracy) : **클래스가 균형 잡혀 있을 때 적합.**
      - 예: 시험 점수 채점 모델.
    - 정밀도(Precision) : **False Positive가 중요한 경우.**
      - 예: 스팸 이메일 분류 (정상 이메일이 스팸으로 분류되면 안 됨).
    - 재현율(Recall) : **False Negative가 중요한 경우.**
      - 예: 암 진단 (암 환자를 놓치는 일이 없어야 함).
    - F1-Score : **정밀도와 재현율의 균형이 중요할 때.**
      - 예: 자연어 처리에서 키워드 추출.

사이킷런을 이용한 분류 성능 평가 예제

```
# 필요한 라이브러리 불러오기
from sklearn.datasets import load_breast_cancer  # 유방암 데이터셋 로드
from sklearn.model_selection import train_test_split  # 학습용/테스트용 데이터 분리
from sklearn.linear_model import LogisticRegression  # 로지스틱 회귀 모델
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report  # 평가 지표

# 1. 데이터셋 로드 및 분리
data = load_breast_cancer()  # 유방암 데이터셋 로드
X = data.data  # 특징 데이터 (환자의 다양한 세포 정보)
y = data.target  # 레이블 데이터 (0: 음성, 1: 양성)

# 학습용 데이터와 테스트용 데이터로 나누기
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# test_size=0.2: 데이터의 20%는 테스트용으로 사용
# random_state=42: 실행할 때마다 동일한 데이터 분리 결과를 얻기 위해 설정

# 2. 로지스틱 회귀 모델 학습
model = LogisticRegression(max_iter=10000)  # 로지스틱 회귀 모델 생성 (최대 반복 횟수 설정)
model.fit(X_train, y_train)  # 학습 데이터로 모델 훈련

# 3. 테스트 데이터 예측
y_pred = model.predict(X_test)  # 테스트 데이터를 사용하여 결과 예측
```

```
# 4. 평가 지표 계산
accuracy = accuracy_score(y_test, y_pred)  # 정확도 계산
precision = precision_score(y_test, y_pred)  # 정밀도 계산
recall = recall_score(y_test, y_pred)  # 재현율 계산
f1 = f1_score(y_test, y_pred)  # F1-Score 계산

# 5. 결과 출력
print("정확도(Accuracy):", accuracy)  # 정확도 출력
print("정밀도(Precision):", precision)  # 정밀도 출력
print("재현율(Recall):", recall)  # 재현율 출력
print("F1-Score:", f1)  # F1-Score 출력

# 혼동 행렬 출력
print("\n혼동 행렬 (Confusion Matrix):\n", confusion_matrix(y_test, y_pred))
# 혼동 행렬을 통해 TP, TN, FP, FN 값 확인

# 분류 보고서 출력
print("\n분류 보고서 (Classification Report):\n",
classification_report(y_test, y_pred))
# Classification Report로 각 클래스에 대한 평가 지표 확인
```

사이킷런을 이용한 분류 성능 평가 예제

```
정확도(Accuracy): 0.956140350877193
정밀도(Precision): 0.9459459459459459
재현율(Recall): 0.9859154929577465
F1-Score: 0.9655172413793104

혼동 행렬 (Confusion Matrix):
 [[39  4]
 [ 1 70]]

분류 보고서 (Classification Report):
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

- **지표 값 출력**
  - 정확도 : 모델이 전체 데이터 중 95%를 정확히 예측.
  - 정밀도 : 양성으로 예측한 데이터 중 94%가 실제로 양성.
  - 재현율 : 실제 양성 데이터 중 98%를 정확히 양성으로 예측.
  - F1-Score : 정밀도와 재현율의 조화 평균(균형).
- **혼동 행렬 출력**
  - 39: 실제 음성 데이터를 음성으로 예측 (TN).
  - 4 : 실제 음성 데이터를 양성으로 잘못 예측 (FP).
  - 1 : 실제 양성 데이터를 음성으로 잘못 예측 (FN).
  - 70: 실제 양성 데이터를 양성으로 예측 (TP).
- **분류 보고서 출력**
  - 클래스 0(음성)
    정밀도 97%, 재현율 91%, F1-Score 94%.
  - 클래스 1(양성)
    정밀도 95%, 재현율 99%, F1-Score 97%.
  - 매크로 평균 : 각 클래스의 단순 평균.
  - 가중 평균 : 클래스 비율에 따라 가중치를 둔 평균.

first coding

모델 성능 향상 - K-최근접 이웃 (KNN)

KNN 알고리즘의 이해

K 값의 선택과 영향

모델의 성능 향상 및 하이퍼파라미터 튜닝

KNN 알고리즘의 이해

- **K-최근접 이웃(KNN) 알고리즘**

  - KNN은 지도 학습(Supervised Learning) 알고리즘 중 하나
  - KNN의 핵심 아이디어는 **가까운 데이터가 비슷한 특성을 가진다**는 가정

    → 새로운 데이터 포인트가 주어졌을 때,

    이를 가장 가까운 데이터 포인트들의 그룹(이웃)으로 분류하거나 값을 예측하는 데 사용
  - KNN은 간단하지만 매우 효과적인 분류(Classification)와 회귀(Regression) 알고리즘

  - KNN을 언제 사용 하면 좋을까?
    - 데이터의 패턴이 비선형적이고 복잡할 때 KNN이 좋은 성능을 보일 수 있.

      단, 데이터가 많거나 고차원일 경우 성능 저하를 겪을 수 있으므로 이런 경우 차원 축소나 데이터 전처리가 필요

KNN 알고리즘의 이해

- KNN의 작동 원리

① 훈련 데이터 준비

- 데이터가 여러 개의 특성(feature)으로 구성되어 있다고 가정
- 각각의 데이터는 특정 레이블(label, 정답) 또는 값을 가지고 있음

② 거리 계산

- 새로운 데이터 포인트(예측하려는 데이터)와 훈련 데이터 사이의 거리를 계산
  → 가장 흔히 사용하는 거리는 **유클리드 거리(Euclidean Distance)**
- 유클리드 거리 공식

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

여기서 xi 와 yi 는 각각 두 점의 i-번째 특성 값

KNN 알고리즘의 이해

- KNN의 작동 원리

③ K개의 이웃 선택

- 가장 가까운 K개의 데이터 포인트를 선택
- K는 미리 정해진 하이퍼파라미터로, K의 값에 따라 모델의 성능이 달라짐

④ 결과 결정

- 분류(Classification)
  - 선택된 K개의 이웃 중 가장 많은 빈도로 나타나는 레이블을 선택
    → 이를 다수결(Majority Voting)이라고 함
- 회귀(Regression)
  - 선택된 K개의 이웃의 값을 평균을 내어 예측 값을 결정

KNN 알고리즘의 이해

- KNN의 주요 특징

  - 모델 학습이 필요하지 않음
    - KNN은 훈련 과정에서 데이터를 저장하기만 하고, 새로운 데이터가 들어올 때 실시간으로 계산
    - 이 때문에 Lazy Learning(게으른 학습) 알고리즘이라고도 함
  - 단순성
    - 복잡한 수학적 계산 없이 데이터 간 거리를 비교하여 예측을 수행하므로 이해하고 구현하기
  - 다양한 응용
    - 분류와 회귀 문제 모두에 사용할 수 있음
    - 예시
      - 분류 : 이메일 스팸 탐지, 질병 진단
      - 회귀 : 주택 가격 예측

사이킷런 기반 KNN 예제

```python
# 1. 필요한 라이브러리 임포트
from sklearn.datasets import load_iris  # Iris 데이터셋 로드
from sklearn.model_selection import train_test_split  # 데이터 분리
from sklearn.neighbors import KNeighborsClassifier  # KNN 알고리즘 사용
from sklearn.metrics import classification_report, accuracy_score  #성능평가

# 2. 데이터 로드 및 확인
iris = load_iris()  # Iris 데이터셋 로드
X = iris.data  # 특성 데이터 (꽃잎/꽃받침의 길이와 너비)
y = iris.target  # 레이블 데이터 (품종: Setosa, Versicolor, Virginica)

# 데이터 구조 확인
print(f"특성 데이터 샘플:\n{X[:5]}")  # 처음 5개의 샘플 출력
print(f"레이블 데이터 샘플:\n{y[:5]}")  # 처음 5개의 레이블 출력

# 3. 데이터 분리 (훈련 세트와 테스트 세트)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 데이터 분리 결과 확인
print(f"훈련 데이터 크기: {X_train.shape}")
print(f"테스트 데이터 크기: {X_test.shape}")
```

```python
# 4. KNN 모델 생성
k = 3  # 가장 가까운 이웃의 개수
knn = KNeighborsClassifier(n_neighbors=k)  # KNN 분류기 생성

# 5. 모델 학습
knn.fit(X_train, y_train)  # 훈련 데이터로 모델 학습

# 6. 모델 평가
y_pred = knn.predict(X_test)  # 테스트 데이터 예측
accuracy = accuracy_score(y_test, y_pred)  # 정확도 계산
print(f"KNN 모델 정확도 (K={k}): {accuracy * 100:.2f}%")

# 분류 성능 세부 보고서 출력
print("\n분류 보고서:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# 7. 새로운 데이터 예측
new_data = [[5.0, 3.5, 1.3, 0.3]]  # 예측할 새로운 데이터 (꽃잎/꽃받침 크기)
prediction = knn.predict(new_data)  # 예측
predicted_class = iris.target_names[prediction][0]  # 예측된 품종 이름
print(f"\n새로운 데이터 {new_data}의 예측 결과: {predicted_class}")
```

사이킷런 기반 KNN 예제

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
레이블 데이터 샘플:
[0 0 0 0 0]
훈련 데이터 크기: (120, 4)
테스트 데이터 크기: (30, 4)
KNN 모델 정확도 (K=3): 100.00%

분류 보고서:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30


새로운 데이터 [[5.0, 3.5, 1.3, 0.3]]의 예측 결과: setosa
```

- **데이터 요약훈련**
  - 데이터
    : 120개 샘플, 4개의 특징
    (꽃받침 길이/너비, 꽃잎 길이/너비).
  - 테스트 데이터 : 30개 샘플.
  - 레이블
    : [0, 1, 2] 각각 setosa, versicolor, virginica를 나타냄.
- **KNN 모델 성능**
  - 정확도 (Accuracy)
    : 100% (테스트 데이터 30개 모두 정확히 분류).
- **분류 보고서 출력**
  - Precision (정밀도): 모든 클래스 1.00.
  - Recall (재현율): 모든 클래스 1.00.
  - F1-Score: 모든 클래스 1.00.
  - Support:setosa: 10개, versicolor: 9개, virginica: 11개.
- **새로운 데이터 예측**
  - 입력 데이터를 setosa로 정확히 예측.

K 값의 선택과 영향

- **K 값의 선택과 영향**

  - KNN 알고리즘에서 K 값은 매우 중요한 하이퍼파라미터
    - K 는 모델이 예측을 할 때 고려하는 가장 가까운 이웃의 수를 의미
    - K 값의 선택은 모델의 성능에 큰 영향을 미침
    - K 는 새로운 데이터 포인트가 주어졌을 때,
      이 데이터와 가장 가까운 K개의 데이터 포인트를 기반으로 예측을 수행
      - 예

        K=1: 가장 가까운 하나의 이웃만 보고 결정.

        K=3: 가장 가까운 3개의 이웃을 보고 다수결로 결정.

K 값의 선택과 영향

- **K 값의 선택과 영향**

  - K 값이 작을 때의 영향
    - 장점
      - 모델이 데이터의 세부적인 패턴을 잘 잡아냄
      - 국소적(local) 특징에 민감하므로 복잡한 데이터를 처리하는 데 적합
    - 단점
      - 과적합(Overfitting) 위험이 증가
      - 노이즈(잘못된 데이터)에 의해 결과가 영향을 받을 가능성이 큼
      - 작은 데이터 변화에도 모델의 결과가 달라질 수 있음

K 값의 선택과 영향

- **K 값의 선택과 영향**

  - K 값 선택 시 고려 사항
    - **데이터의 크기**
      - 작은 데이터셋 : K 값을 **작게 설정**하는 것이 적합. 예: 데이터가 50개라면 K=3 또는 K=5
      - 큰 데이터셋: K 값을 **더 크게 설정**해도 됨. 예: 데이터가 수천 개라면 K=15 또는 K=20
    - **홀수 값의 선택**
      - **K는 보통 홀수로 설정하는 것이 좋음** → 이는 동률(Tie)을 방지하기 위함
      - 클래스가 2개일 때 K가 짝수라면 결과가 모호해질 수 있음
    - **성능 평가를 위한 테스트교차 검증(Cross Validation)**
      - 다양한 K 값을 시도하면서 모델 성능(정확도)을 비교하여 최적의 K 값을 선택
      - 과적합과 과소적합 사이에서 적절한 균형을 찾을 수 있음

K 값 선택 실습 코드 (사이킷런)

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# 1. 데이터 로드 및 분리
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.2, random_state=42)

# 2. K 값에 따른 정확도 비교
accuracies = {}  # K 값별 정확도를 저장할 딕셔너리

for k in range(1, 21):  # K 값을 1부터 20까지 테스트
    knn = KNeighborsClassifier(n_neighbors=k)  # KNN 모델 생성
    knn.fit(X_train, y_train)  # 모델 학습
    y_pred = knn.predict(X_test)  # 예측
    accuracy = accuracy_score(y_test, y_pred)  # 정확도 계산
    accuracies[k] = accuracy  # K 값별 정확도 저장

# 3. 결과 출력
for k, acc in accuracies.items():
    print(f"K={k}: 정확도={acc * 100:.2f}%")
```

```python
# 4. 최적의 K 값 찾기
best_k = max(accuracies, key=accuracies.get)  # 정확도가 가장 높은 K 값
print(f"\n최적의 K 값: {best_k}, 정확도: {accuracies[best_k] * 100:.2f}%")
```

```
K=1: 정확도=100.00%
K=2: 정확도=100.00%
K=3: 정확도=100.00%
K=4: 정확도=100.00%
K=5: 정확도=100.00%
K=6: 정확도=100.00%
K=7: 정확도=96.67%
K=8: 정확도=100.00%
K=9: 정확도=100.00%
K=10: 정확도=100.00%
K=11: 정확도=100.00%
K=12: 정확도=100.00%
K=13: 정확도=100.00%
K=14: 정확도=100.00%
K=15: 정확도=100.00%
K=16: 정확도=100.00%
K=17: 정확도=100.00%
K=18: 정확도=100.00%
K=19: 정확도=100.00%
K=20: 정확도=100.00%
```

모델의 성능 향상 및 하이퍼파라미터 튜닝

- **모델의 성능 향상 및 하이퍼파라미터 튜닝**

  - KNN 알고리즘의 성능은 다양한 요소에 의해 영향을 받음
    → 성능을 최적화하려면 데이터 전처리, 거리 계산 방식, 그리고 하이퍼파라미터(특히 K 값과 거리 계산 방식)를 신중히 조정해야 함
  - **모델 성능 향상 방법**
    - **데이터 전처리**
      스케일링, 이상치 제거, 특성 선택.
    - **거리 계산 방식**
      유클리드, 맨해튼, 민코프스키 등을 데이터 특성에 맞게 선택.
    - **하이퍼파라미터 튜닝**
      최적의 K 값을 교차 검증으로 선택. 가중치를 부여하여 더 나은 성능 도출.

모델의 성능 향상 및 하이퍼파라미터 튜닝 예제

```python
# 1. 필요한 라이브러리 임포트
from sklearn.datasets import load_iris  # Iris 데이터셋 로드
from sklearn.model_selection import train_test_split, cross_val_score  # 데이터 분리 및 교차 검증
from sklearn.neighbors import KNeighborsClassifier  # KNN 알고리즘 사용
from sklearn.preprocessing import StandardScaler  # 데이터 스케일링
from sklearn.metrics import accuracy_score, classification_report  #성능평가

# 2. 데이터 로드 및 분리
iris = load_iris()  # Iris 데이터셋 로드
X = iris.data  # 특성 데이터 (꽃잎/꽃받침의 길이와 너비)
y = iris.target  # 레이블 데이터 (품종: Setosa, Versicolor, Virginica)

# 훈련 세트와 테스트 세트로 데이터 분리 (80% 훈련, 20% 테스트)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. 데이터 스케일링 (KNN은 거리 기반 알고리즘이므로 중요)
scaler = StandardScaler()  # 스케일러 생성
X_train_scaled = scaler.fit_transform(X_train)  # 훈련 데이터 스케일링
X_test_scaled = scaler.transform(X_test)  # 테스트 데이터 스케일링

# 4. K 값 튜닝 (교차 검증 사용)
best_k = 1  # 최적의 K 값 초기화
best_score = 0  # 최적의 정확도 초기화
```

```python
# 다양한 K 값을 테스트하며 최적의 K 값 찾기
for k in range(1, 21):  # K=1부터 K=20까지 테스트
    knn = KNeighborsClassifier(n_neighbors=k)  # KNN 모델 생성
    scores = cross_val_score(knn, X_train_scaled, y_train, cv=5) # 5-fold 교차 검증
    mean_score = scores.mean()  # 교차 검증 정확도의 평균
    if mean_score > best_score:  # 가장 높은 정확도를 갱신
        best_k = k
        best_score = mean_score

print(f"최적의 K 값: {best_k}, 교차 검증 정확도: {best_score * 100:.2f}%")
# 5. 거리 계산 방식 변경 및 가중치 추가
# 최적의 K 값과 다른 거리 계산 방식을 사용하여 모델 생성
final_knn = KNeighborsClassifier(n_neighbors=best_k, weights='distance', metric='manhattan')
final_knn.fit(X_train_scaled, y_train)  # 최적의 하이퍼파라미터로 모델 학습
# 6. 최적화된 모델 평가
y_pred = final_knn.predict(X_test_scaled)  # 테스트 데이터로 예측
accuracy = accuracy_score(y_test, y_pred)  # 정확도 계산
print(f"최적화된 KNN 모델 정확도: {accuracy * 100:.2f}%")
# 세부적인 성능 보고서 출력
print("\n분류 보고서:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
# 7. 새로운 데이터 예측
new_data = [[5.0, 3.5, 1.3, 0.3]]  # 예측할 새로운 데이터 (꽃잎/꽃받침 크기)
new_data_scaled = scaler.transform(new_data)  # 새로운 데이터 스케일링
prediction = final_knn.predict(new_data_scaled)  # 예측
predicted_class = iris.target_names[prediction][0]  # 예측된 품종 이름
print(f"\n새로운 데이터 {new_data}의 예측 결과: {predicted_class}")
```

first coding

실습 프로젝트 - 간단한 지도학습 모델 만들기

데이터 준비 → 모델 학습 → 성능 평가의 전 과정을 실습

학생들이 간단한 데이터를 직접 다뤄보며 이해도 점검