

# 파이썬 프로그래밍

## 20. 딥러닝 파이썬 패키지(2)

## ❖ 수업 목표

- statsmodels 모듈의 기본 함수를 사용할 수 있다.
- scikit-learn 모듈의 기본 함수를 사용할 수 있다.
- scipy 모듈의 기본 함수를 사용할 수 있다.

## ❖ 세부 목표

- 20.1 statsmodels 패키지
- 20.2 scikit-learn 개요
- 20.3 scikit-learn 활용
- 20.4 scipy 패키지

## ❖ statsmodels 패키지

- 통계적 모델 추정 및 통계적 테스트 실시 및 통계적 데이터 탐색을 위한 클래스와 함수를 제공
- R 스타일 공식과 pandas 데이터프레임을 사용한 모델 지정 지원

### ■ 주요 기능

- 선형 및 로지스틱 회귀분석
  - 데이터의 종속 변수와 한 개 이상의 독립 변수 간의 관계를 모델링
- 시계열 분석
  - ARIMA(자기회귀 누적 이동 평균) 모델을 포함한 시계열 분석을 위한 모델 제공
  - 시계열: 시간의 흐름에 따라 기록된 것
- 통계적 테스트
  - 가설 검정(검사 및 결정), 모델 선택, 변수 선택을 위한 테스트 제공
- 데이터 탐색 및 시각화
  - 데이터의 기초 통계량 계산 및 시각화를 통한 데이터 패턴 탐색

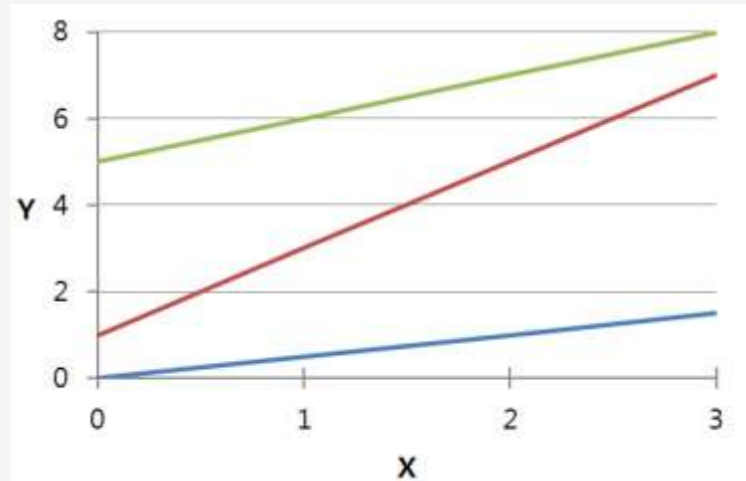
## ❖ 선형 회귀(Linear regression)

- 레이블이 포함된 데이터를 통해 생성한 모델에 새로운 데이터를 입력하고 결과 값을 예측하는 것
  - 회귀: 평균보다 크거나 작은 값들이 평균으로 돌아간다는 가정을 이용한 분석
- 독립 변수와 종속 변수 간의 선형 관계를 모델링하는 기법
  - 예를 들어, 시험공부를 5시간 한 학생이 50점 받았고, 10시간 한 학생이 100점을 받았다면, 7시간 한 학생은 70점의 결과가 예측된다는 것
- 우리가 살고 있는 세상에 많은 데이터나 현상이 선의 형태로 설명 되는 것이 많다는 점에서 매우 유용

# 1. statsmodels 패키지

## ❖ 선형 회귀(Linear regression)

- 선형 회귀에서는 선의 형태가 어떻게 이루어 지는지 찾는 것이 학습을 의미
- 일반적으로 다음과 같은 방정식으로 표현
  - $H(x) = Wx + b$ 
    - $H(x)$ 는 가설(Hypothesis)을 의미하며, 선의 모양은 기울기  $W$ 와 Y절편  $b$ 에 따라 달라지며 1차 방정식의 형태가 됨



- 가설로 생성한 1차 방정식 그래프가 입력 데이터와 일치한다면 좋은 가설이 됨
- 하지만 일치하지 않는다면 그래프의 모양을 결정하는  $W$ 와  $b$ 를 변경하여 데이터와 가장 근접하도록 모델을 변경하는 것(가설을 세우는 것)이 필요

# 1. statsmodels 패키지

## ❖ statsmodels 기본 사용법

### ■ 설치

- `pip install statsmodels`

### ■ импорт

- `import statsmodels.api as sm`

## ❖ 데이터 준비 및 탐색

### ■ 예제 데이터 셋 로드

- r 데이터 셋 다운로드 및 반환
  - `data = sm.datasets.get_rdataset("mtcars").data`
  - `print(data.head())`

### ■ mtcars 데이터 셋

- 1974년 모터트랜드 US 매거진에서 가져온 데이터
- 32종 자동차의 10가지 디자인과 성능특성과 연료소모량
  - `mpg`(연비), `cyl`(실린더수), `disp`(배기량), `hp`(마력) 등

## ❖ 선형 회귀 분석

### ■ 단순 선형 회귀

- 데이터 준비

- `X = data['hp']` # 마력
- `y = data['mpg']` # 연비

- 상수항 추가

- `X = sm.add_constant(X)`
  - » `sm.add_constant()`: 독립 변수에 상수항(절편)을 추가

- 모델 생성 및 학습(`fit`)

- » OLS(Ordinary Least Squares) : 오차의 제곱을 최소화하는 기울기와 절편을 추정하는 방식
- `model = sm.OLS(y, X).fit()`
  - » `sm.OLS()`: 선형 회귀 모델을 생성
  - » `fit()`: 모델을 학습

### ■ 결과 출력

- `print(model.summary())`
  - » `summary()`: 모델 결과를 요약하여 출력

## ❖ 결과 분석

### ■ OLS Regression Results

- No.observations (Number of observations)
  - 총 표본 수
- DF Residuals
  - 잔차(표본으로 추정된 회귀식과 실제 관측값의 차이)의 자유도
  - 전체 표본 수에서 측정되는 변수들(종속변수 및 독립변수)의 개수를 뺀 것
- DF Model
  - 독립변수의 개수

## ❖ 결과 분석

### ■ OLS Regression Results

- R-squared
  - R 제곱이라는 뜻으로 결정 계수
  - 회귀 모델에서 독립변수가 종속변수를 얼마만큼 설명해주는지 가리키는 지표
    - » 예를 들어, 0.6(60%)라고 하면 독립변수가 종속변수의 60% 정도를 설명하는 것을 의미
  - 독립변수의 수가 증가하면 상승하는 경향이 있음
  - 범위는 0에서 1 사이의 값으로 0이면 모델의 설명력이 전혀 없는 상태이고, 1에 가까울수록 모델이 데이터를 잘 설명해주는 상태
- Adj. R-squared
  - 수정된 R 제곱
  - 독립변수의 개수가 많아지면 결정계수의 값이 증가하는 문제를 해결하기 위해 독립변수의 개수와 표본 크기를 고려하여 결정계수를 조정한 값

## ❖ 결과 분석

### ■ OLS Regression Results

- F-statistic
  - F 검정. 전체 회귀모델이 통계적으로 유의한지 검정하기 위해 사용되는 값
  - 회귀모델의 분산과 오차항의 분산의 비율로 계산
  - F값이 크면 회귀모델이 통계적으로 유의하다고 판단
  - F값은 회귀모델의 전체적인 유의성을 평가하며, 반면 각 독립변수의 개별적인 영향력을 t-통계량을 사용하여 검정
- Prob (F-statistic)
  - F-통계량에 대한 P-값(value)
  - 회귀분석 모델 전체에 대한 통계적 의미 판단(0.05이하인 경우 독립변수 유의미)
- AIC
  - 표본의 개수와 모델의 복잡성을 기반으로 모델을 평가하며, 수치가 낮을 수록 좋음
- BIC
  - AIC와 유사하나 패널티를 부여하여 AIC보다 모델 평가 성능이 더 좋으며, 수치가 낮을 수록 좋음

## ❖ 결과 분석

### ■ OLS Regression Results

- **coef(coefficient, 회귀계수)**
  - 데이터로부터 얻은 계수(절편, 기울기)의 추정치
    - » 예를 들어, 출력 값에서 coef 의 const가 30.0989이고 hp가 -0.0682 이면,
    - » 선형 회귀분석 모델 식에 넣어보면,  $H(x) = -0.0682 * x + 30.0989$
    - » 이는 x가 1 상승할 때마다 H(x)가 0.0682 감소한다고 해석 가능
- **std err**
  - 계수 추정치의 표준오차(standard error), 값이 작을 수록 좋음
- **t**
  - t-검정(test), 독립변수와 종속변수 사이의 상관관계, 값이 클 수록 상관도가 큼
    - » t는 회귀 계수가 0이라는 가설 하에서 관측된 계수의 표준 오차 대비 크기
- **P>|t|**
  - p-값(value)
  - 회귀 계수가 통계적으로 유의미한지 표시 (0.05보다 작을 경우 유의미)
    - » P는 독립 변수가 종속 변수에 유의미한 영향을 미칠 확률

## ❖ 선형 회귀 분석

### ■ 다중 선형 회귀

#### • 데이터 준비

- `X = data[['hp', 'wt']]`                      # 마력, 중량
- `y = data['mpg']`                              # 연비
  - » 독립 변수가 여러 개인 경우, [['hp', 'wt']]와 같이 선택

#### • 상수항 추가

- `X = sm.add_constant(X)`
  - » `sm.add_constant()`: 독립 변수에 상수항(절편)을 추가

#### • 모델 생성 및 학습(fit)

- `model = sm.OLS(y, X).fit()`
  - » `sm.OLS()`: 선형 회귀 모델을 생성 / `fit()`: 모델을 학습

#### • 결과 출력

- `print(model.summary())`
  - » `summary()`: 모델 결과를 요약하여 출력

## 2. scikit-learn 패키지 개요

### ❖ Scikit-learn 소개

- Python 기반의 머신러닝 라이브러리
- 다양한 데이터 전처리, 모델링, 평가 도구 제공

### ■ Scikit-learn 주요 특징

- 간단하고 일관된 API
  - 공통적 메서드 제공: `fit(X, y)`, `predict(X)`, `score(X, y)`
  - 통일된 입/출력 형식: Numpy 배열, Pandas 데이터프레임 등 표준화된 데이터 구조
- 풍부한 알고리즘 지원 (분류, 회귀, 클러스터링 등)
- 확장성과 다양한 데이터셋 제공

### ■ 설치 및 환경 설정

- `pip install scikit-learn`
- **임포트**
  - `import sklearn as sk`

### ❖ XOR 연산 학습

- 배타적 논리합(XOR)
- 두 입력 중 하나만 참이고, 다른 한 쪽이 거짓일 때 참
- 모두 참이거나 모두 거짓인 경우는 거짓

P	Q	P xor Q
0	0	0
1	0	1
0	1	1
1	1	0

## 2. scikit-learn 패키지 개요

### ❖ XOR 연산 학습

#### ■ 데이터 직접 입력하여 학습하기

- xor.py

```
from sklearn import svm
from sklearn import metrics

# 데이터 학습하기
clf = svm.SVC() # 알고리즘 선택
# fit(학습데이터, 학습레이블)
clf.fit([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
],
[0, 1, 1, 0]) # 학습(모델링)
```

## 2. scikit-learn 패키지 개요

### ❖ XOR 연산 학습

#### ■ 데이터 직접 입력하여 학습하기

- xor.py

```
# 데이터 예측하기
# predict(테스트데이터)
pre = clf.predict([
    [0, 1],      # 1
    [1, 1]       # 0
])              # 예측 => 답
print(pre)

# 결과 확인
ac_score = metrics.accuracy_score([1, 0], pre)
print(f"정답률: {ac_score}")
```

- 실행 결과

» [1 0]  
» 정답률: 1.0

## 2. scikit-learn 패키지 개요

### ❖ XOR 연산 학습

#### ■ 데이터 셋 학습 데이터와 레이블 분할 후 학습하기

- xor-train2.py

```
from sklearn import svm          # 알고리즘 관련
from sklearn import metrics      # 행렬, 정답률 관련
import pandas as pd              # 데이터 조작 관련
# 데이터 준비(입력) - 지도 학습(데이터/정답)
xor_data=[
    # P, Q, result
    [0, 0, 0],
    [0, 1, 1],
    [1, 0, 1],
    [1, 1, 0]
]
# 데이터 가공 - 학습을 위해 데이터와 레이블 분리
xor_df = pd.DataFrame(xor_data)    # 매개변수: 입력 데이터
# xor_df.loc[전체데이터, 추출할데이터]
# 학습 데이터
data = xor_df.loc[:, 0:1]
# 레이블(정답)
label = xor_df.loc[:, 2]
```

## 2. scikit-learn 패키지 개요

### ❖ XOR 연산 학습

#### ■ 데이터 셋 학습 데이터와 레이블 분할 후 학습하기

- xor-train2.py

```
# 학습 알고리즘 - 분류(Classification)
# 모델 생성
clf = svm.SVC()
clf.fit(data, label)    # 학습 : 1.학습데이터 / 2.정답

# 답 예측
pre = clf.predict(data)
print("예측결과:", pre)
# 정답률 확인
ac_score = metrics.accuracy_score(label, pre)
print(f"정답률= {ac_score}")
```

- 실행 결과

- » 예측결과: [0 1 1 0]
- » 정답률= 1.0

### 3. scikit-learn 패키지 활용

#### ❖ 붓꽃의 품종 분류하기

- 붓꽃은 150종류 이상의 품종
- 일반적인 사람은 물론 꽃을 조금 아는 사람도 품종을 분류하기 어려움
- 머신러닝을 사용해 꽃잎과 꽃받침의 크기를 기반으로 분류



### ❖ 붓꽃의 품종 분류하기

#### ■ 붓꽃 데이터 구하기

- “Fisher의 붓꽃 데이터”라는 유명한 붓꽃 분류 데이터가 공개되어 있음
- **sklearn** 패키지 내 **datasets** 모듈을 통해 로드 가능
- 예제 데이터셋 로드 : **iris.py**

```
from sklearn.datasets import load_iris

# 1. Iris 데이터셋 로드
iris = load_iris()
```

## 3. scikit-learn 패키지 활용

### ❖ 붓꽃의 품종 분류하기

#### ■ 붓꽃 데이터 구하기

- iris.py

```
import pandas as pd

df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

# 0, 1, 2 로 표현된 label을 문자열로 매핑
target_names = {
    0:iris.target_names[0],      # 'setosa'
    1:iris.target_names[1],      # 'versicolor'
    2:iris.target_names[2]       # 'virginica'
}

df['target'] = df['target'].map(target_names)

print(df.head())
```

#### • 실행 결과

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
• 0	5.1	3.5	1.4	0.2	0
• 1	4.9	3.0	1.4	0.2	0
• 2	4.7	3.2	1.3	0.2	0
• 3	4.6	3.1	1.5	0.2	0
• 4	5.0	3.6	1.4	0.2	0

### ❖ 붓꽃의 품종 분류하기

#### ■ 훈련 데이터와 시험 데이터로 분할하기

- sklearn 패키지 내 model\_selection 모듈로 부터 데이터 분할 함수 사용
- iris.py

```
from sklearn.model_selection import train_test_split

# 2. 필요한 열 추출하기
iris_data = df[[
    "sepal length (cm)",
    "sepal width (cm)",
    "petal length (cm)",
    "petal width (cm)"
]]
iris_label = df["target"]

# 3. 학습 전용 데이터와 테스트 전용 데이터로 나누기
train_data, test_data, train_label, test_label = \
    train_test_split(iris_data, iris_label)
```

### ❖ 붓꽃의 품종 분류하기

#### ■ 데이터 학습 및 예측하기

- sklearn 패키지 내 분류 알고리즘 svm 모듈로부터 SVC 클래스 사용
- svm : 서포트 벡터 머신(Support Vector Machine)
  - SVC : 서포트 벡터 분류(Support Vector Classification)
    - 데이터 집합 내에서 최대 마진을 가진 최적의 선(평면)을 찾아 데이터 분류
  - fit() : 주어진 훈련 데이터에 따라 모델을 피팅
  - predict() : 시험 데이터에 대해 분류를 수행
- iris.py

```
from sklearn import svm, metrics

# 4. 데이터 학습시키고 예측하기
clf = svm.SVC()
clf.fit(train_data, train_label)
pre = clf.predict(test_data)
```

### ❖ 붓꽃의 품종 분류하기

#### ■ 정답률 구하기

- sklearn 패키지 metrics 모듈로부터 accuracy\_score 함수 사용
- metrics : (성능) 지표
  - accuracy\_score() : 정확도 분류 점수(정답률)

- iris.py

```
# 5. 정답률 구하기
ac_score = metrics.accuracy_score(test_label, pre)
print("정답률 =", ac_score)
```

- 실행 결과
  - 정답률 = 0.9473684210526315

## 4. scipy 패키지

### ❖ scipy(사이파이) 소개

- Python의 과학 및 공학 계산 라이브러리
- NumPy와 함께 사용되어 데이터 분석 및 수학적 계산에 적합
- 주요 서브모듈: linalg(선형대수), optimize(최적화), stats(통계), integrate(적분), sparse(희소행렬)

### ■ Scipy와 NumPy의 차이점

- NumPy: 다차원 배열과 기본 연산 제공
- Scipy: 고급 수학 및 과학 계산 기능 제공 (NumPy 위에 구축됨)

### ■ 설치 및 환경 설정

- `pip install scipy`
- **임포트**
  - `import scipy as sp`

### ❖ Scipy 주요 모듈과 기능

#### ■ 선형 대수(linear algebra) 계산 (scipy.linalg)

##### • 선형 방정식 풀기

- `from scipy.linalg import solve`
- `# Ax = b 형태의 선형 방정식`
- `# 3x + 1y = 9`
- `# 1x + 2y = 8`
- `A = [[3, 1], [1, 2]]`
- `b = [9, 8]`
- `x = solve(A, b)`
- `print(f"Solution: {x}")`

### ❖ Scipy 주요 모듈과 기능

#### ■ 최적화 (scipy.optimize)

- 함수를 최소화하는 변수 값 구하기
  - `from scipy.optimize import minimize`
  - `# 단순 2차 함수 최적화`
  - `def f(x):`
  - `return x**2 + 4*x + 4`
  - `# x0는 함수의 독립변수(x)의 초기 값(시작점)`
  - `result = minimize(f, x0=0)`
  - `print(f"Optimal value: {result.x}")`

### ❖ Scipy 주요 모듈과 기능

#### ■ 최적화 (scipy.optimize)

##### • 방정식의 근 찾기

- `from scipy.optimize import root`
- `# 방정식  $x^2 - 4 = 0$ 의 근`
- `def equation(x):`
- `return x**2 - 4`
- `sol = root(equation, x0=1)`
- `print(f"Root: {sol.x}")`

## 4. scipy 패키지

### ❖ Scipy 주요 모듈과 기능

#### ■ 통계 (scipy.stats)

- 기본 통계 계산 (최소, 최대, 평균, 분산, 외도, 첨도)

- `from scipy.stats import describe`

- `data = [1, 2, 3, 4, 5, 6, 7]`

- `stats = describe(data)`

- `print(stats)`

- 실행 결과

- » `DescribeResult(nobs=7, minmax=(np.int64(1), np.int64(7)), mean=np.float64(4.0), variance=np.float64(4.666666666666667), skewness=np.float64(0.0), kurtosis=np.float64(-1.25))`

### ❖ Scipy 주요 모듈과 기능

#### ■ 희소 행렬 (scipy. sparse)

- 0이 많은 2차원 데이터를 0값을 가지지 않는 요소만 표시하는 COO(Coordinate) 포맷으로 변경
  - `from scipy.sparse import csr_matrix`
  - `data = [`
    - `[0, 0, 3],`
    - `[4, 0, 0],`
    - `[0, 5, 0]`
  - `]`
  - `sparse_matrix = csr_matrix(data)`
  - `print(sparse_matrix)`
  - 실행 결과
    - » `<Compressed Sparse Row sparse matrix of dtype 'int64'`
    - » `with 3 stored elements and shape (3, 3)>`
    - » 

Coords	Values
(0, 2)	3
(1, 0)	4
(2, 1)	5

## ❖ 과제

- 1. statsmodels 패키지의 기본 함수 사용하여 코드 작성하기
- 2. scikit-learn 패키지의 기본 함수를 사용하여 코드 작성하기
- 3. scipy 패키지의 기본 함수를 사용하여 코드 생성하기

## ❖ 다음 수업 내용

- 딥러닝 파이썬 패키지(3)
  - seaborn
  - opencv