

01-2

코랩과 주피터 노트북

핵심 키워드

코랩

노트북

구글 드라이브

이 책의 모든 코드는 웹 브라우저에서 파이썬 코드를 실행할 수 있는 구글 코랩^{Colab}을 사용하여 작성했습니다. 본격적으로 머신러닝을 배우기 전에 구글 코랩에 대해 소개하고 간단한 사용법을 익히려 합니다.

The screenshot shows the Google Colab web interface. A modal dialog titled '노트 열기' (Open Notebook) is displayed in the center. The dialog has a search bar at the top with the placeholder text '노트북 검색' (Search Notebook). Below the search bar is a table with columns: '제목' (Title), '마지막 연 시간' (Last Modified Time), and '처음 연 시간' (First Modified Time). The table contains one entry: 'Colaboratory에 오신 것을 환영합니다' (Welcome to Colaboratory), with both timestamps set to '오전 10:39' (10:39 AM). To the left of the table is a sidebar with a list of sources: '예' (Example), '최근 사용' (Recently Used), 'Google Drive', 'GitHub', and '업로드' (Upload). The '최근 사용' (Recently Used) option is highlighted with a blue border. At the bottom of the dialog, there is a '+ 새 노트' (New Notebook) button on the left and a '취소' (Cancel) button on the right. The background interface shows the Colab logo, a welcome message, and a sidebar with various navigation options like '시작하기' (Get Started), '데이터 과학' (Data Science), and '머신러닝' (Machine Learning).

Colab 시작

(신규) Gemini

- [Generate a G](#)
- [Talk to Gemin](#)
- [Gemini API: Q](#)
- [Gemini API co](#)
- [Compare Gen](#)
- [More notebo](#)

Colab에 이미 익숙하

[] 코딩을 시작하

Colab이란?

Colaboratory(줄여서

- 구성이 필요하지 않음
- 무료로 GPU 사용
- 간편한 공유

Colaboratory에 오신 것을 환영합니다 → 제목

파일 수정 보기 삽입 런타임 도구 도움말 변경사항을 저장할 수 없음

공유

RAM 디스크

수정 가능

목차

시작하기

데이터 과학

머신러닝

추가 리소스

머신러닝 예시

섹션

시작하기

지금 읽고 계신 문서는 정적 웹페이지가 아니라 코드를 작성하고 실행할 수 있는 대화형 환경인 Colab 메모장입니다.

예를 들어 다음은 값을 계산하여 변수로 저장하고 결과를 출력하는 간단한 Python 스크립트가 포함된 코드 셀입니다.

Colaboratory란?

줄여서 'Colab'이라고도 하는 Colaboratory를 사용하면 브라우저에서 Python을 작성하고 실행할 수 있습니다. Colab은 다음과 같은 이점을 자랑합니다.

- 구성이 필요하지 않음
- GPU 무료 액세스
- 간편한 공유

학생이든, 데이터 과학자든, AI 연구원이든 Colab으로 업무를 더욱 간편하게 처리할 수 있습니다. [Colab 소개 영상](#)에서 자세한 내용을 확인하거나 아래에서 시작해 보세요.

```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

86400

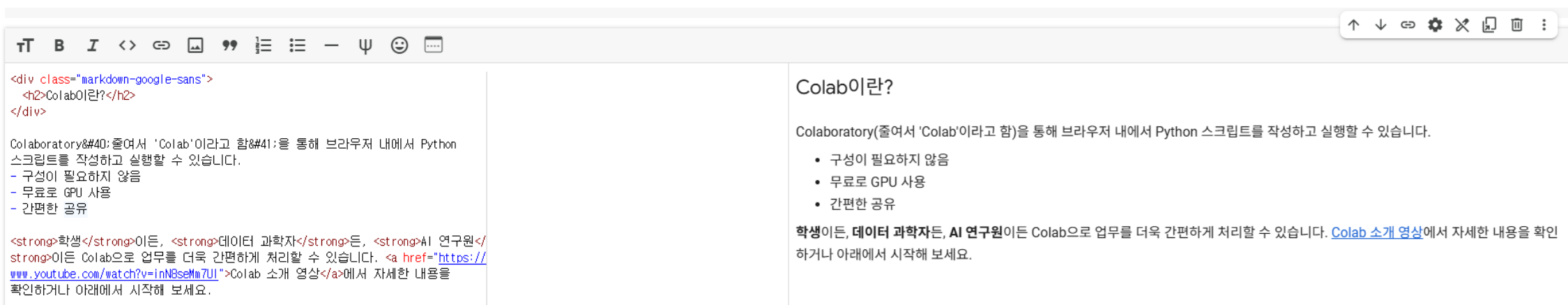
셀을 선택하면 이렇게 그림자가 나타납니다.

텍스트 셀

셀^{cell}은 코랩에서 실행할 수 있는 최소 단위입니다. 즉 셀 안에 있는 내용을 한 번에 실행하고 그 결과를 노트북에 나타냅니다. 하지만 텍스트 셀은 코드처럼 실행되는 것이 아니기 때문에 자유롭게 사용해도 괜찮습니다. 셀 하나에 아주 긴 글을 써도 되고 여러 셀에 나누어 작성해도 괜찮습니다.

텍스트 셀을 수정하려면 원하는 셀로 이동한 후 `Enter` 키를 누르거나 마우스를 더블 클릭하여 편집 화면으로 들어갑니다. 첫 번째 셀을 편집해 봅시다.

텍스트 셀에서는 HTML과 **마크다운** Markdown 을 혼용해서 사용할 수 있습니다. 왼쪽 창에서 텍스트를 수정하면 오른쪽 미리 보기 창에서 수정된 결과를 바로 볼 수 있습니다. <h1> 태그 아래에 임의의 텍스트를 추가하고 미리 보기 창에 나타나는 결과를 확인해 보세요.



Colab이란?


Colaboratory(줄여서 'Colab'이라고 함)을 통해 브라우저 내에서 Python 스크립트를 작성하고 실행할 수 있습니다.


- 구성이 필요하지 않음
- 무료로 GPU 사용
- 간편한 공유
- 재미있다
- 신기하다


코드 셀

‘Colaboratory에 오신 것을 환영합니다’ 노트북의 세 번째 셀이 코드 셀입니다. 코드 셀로 이동하면 코드와 결과가 함께 선택됩니다.

```
seconds_in_a_day = 24 * 60 * 60  
seconds_in_a_day
```

 86400

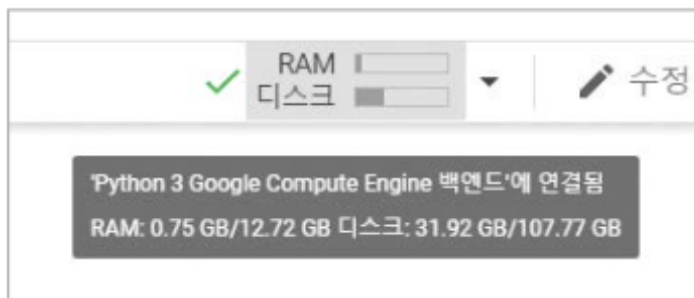
 `seconds_in_a_day = 24 * 60 * 60 * 2 - 10000`
`seconds_in_a_day`

 162800

노트북

코랩은 구글이 대화식 프로그래밍 환경인 주피터^{Jupyter}를 커스터마이징한 것입니다. 파이썬 지원으로 시작한 주피터 프로젝트¹는 최근에는 다른 언어도 지원합니다. 주피터 프로젝트의 대표 제품이 바로 노트북^{Notebook}입니다. 흔히 주피터 노트북이라고 부릅니다.

코랩 노트북은 구글 클라우드의 가상 서버^{Virtual Machine}를 사용합니다. 화면 오른쪽 상단에 있는 RAM, 디스크 아이콘에 마우스를 올리면 상세 정보를 알 수 있습니다. 코드를 실행하기 전이나 연결이 끊어진 상태에서는 아이콘 대신에 [연결] 버튼이 활성화됩니다.



새 노트북 만들기


이제 코랩에서 새로운 노트북을 만들고 저장하겠습니다.

01 [파일]-[새 노트]를 클릭해서 새로운 노트북을 만듭시다.



02 새 노트북은 Untitled[숫자].ipynb 이름으로 만들어지고 노트북에는 다음과 같이 빈 코드의 셀 하나가 들어 있습니다.



03 코드 셀에 'Hello World'를 출력하는 `print()` 코드를 작성하고 이 파일의 이름을 'Hello World'로 저장해 봅시다. 먼저 빈 코드 셀을 마우스로 선택하고 다음과 같이 입력한 다음 셀을 실행해 보세요. 코드 셀을 실행하려면 `Ctrl` + `Enter` 키(macOS는 `cmd` + `Enter` 키)를 누르거나 왼쪽에 있는 플레이 아이콘()을 클릭합니다.



04 노트북은 자동으로 구글 드라이브 Google Drive의 [내 드라이브]-[Colab Notebooks] 폴더 아래 저장됩니다. 웹 브라우저에서 구글 드라이브(<https://drive.google.com>)로 접속해서 확인해 보세요.



note [파일]-[저장]을 선택해 수동으로 노트북을 저장할 수도 있습니다.

- 05 노트북의 이름을 바꿔 보겠습니다. 제목을 마우스로 클릭하면 수정할 수 있도록 바뀝니다. 이 파일의 제목을 'Hello World'로 바꿔 보세요.



마무리

▶ 키워드로 끝내는 핵심 포인트

- **코랩**은 구글 계정이 있으면 누구나 사용할 수 있는 웹 브라우저 기반의 파이썬 코드 실행 환경입니다.
- **노트북**은 코랩의 프로그램 작성 단위이며 일반 프로그램 파일과 달리 대화식으로 프로그램을 만들 수 있기 때문에 데이터 분석이나 교육에 매우 적합합니다. 노트북에는 코드, 코드의 실행 결과, 문서를 모두 저장하여 보관할 수 있습니다.
- **구글 드라이브**는 구글이 제공하는 클라우드 파일 저장 서비스입니다. 코랩에서 만든 노트북은 자동으로 구글 드라이브의 'Colab Notebooks' 폴더에 저장되고 필요할 때 다시 코랩에서 열 수 있습니다.

텍스트 셀에 사용할 수 있는 마크다운

마크다운 형식	설명	예제
# 제목1	<h1> 태그와 동일합니다.	제 목1
## 제목2	<h2> 태그와 동일합니다.	제 목2
### 제목3	<h3> 태그와 동일합니다.	제 목3
#### 제목4	<h4> 태그와 동일합니다.	제 목4
##### 제목5	<h5> 태그와 동일합니다.	제 목5
혼공머신	굵게 씁니다.	혼공신
혼공머신 _혼공머신_	기울임 꼴로 씁니다.	<i>혼공신</i>
~~혼공머신~~	취소선을 추가합니다.	혼공신
`print("Hello World!")`	백틱 기호를 사용해 코드 서체로 씁니다.	<code>print("Hello World!")</code>
> 혼공머신	들여쓰기합니다. 여러 단계를 들여쓸 수 있습니다.	혼공신
* 혼공머신 - 혼공머신	글머리 기호 목록을 만듭니다.	• 혼공신
[한빛미디어](http://www.hanbit.co.kr/)	링크를 만듭니다.	한빛미디어
![한빛미디어](http://www.hanbit.co.kr/images/common/logo_hanbit.png)	이미지를 추가합니다.	 한빛출판네트워크
\$ y = x \times z \$	레이텍을 추가합니다.	$y = x \times z$

▶ 확인 문제

1. 구글에서 제공하는 웹 브라우저 기반의 파이썬 실행 환경은 무엇인가요?

- ① 주피터 노트북
- ② 코랩
- ③ 크롬
- ④ 아나콘다

2. 코랩 노트북에서 쓸 수 있는 마크다운 중에서 다음 중 기울임 꼴로 쓰는 것은?

- ① ****혼공머신****
- ② *~~혼공머신~~*
- ③ ‘혼공머신’
- ④ _혼공머신_

3. 코랩 노트북은 어디에서 실행되나요?

- ① 내 컴퓨터
- ② 구글 드라이브
- ③ 구글 클라우드
- ④ 아마존 웹서비스

01-3

마켓과 머신러닝

핵심 키워드

특성

훈련

k-최근접 이웃 알고리즘

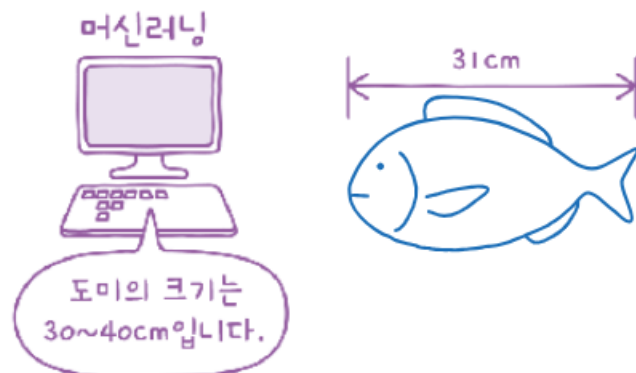
모델

정확도

가장 간단한 머신러닝 알고리즘 중 하나인 k-최근접 이웃을 사용하여 2개의 종류를 분류하는 머신러닝 모델을 훈련합니다.

생선 분류 문제

한빛 마켓에서 팔기 시작한 생선은 ‘도미’, ‘곤들매기’, ‘농어’, ‘강꼬치고기’, ‘로치’, ‘빙어’, ‘송어’입니다. 이 생선들은 물류 센터에 많이 준비되어 있습니다. 이 생선들을 프로그램으로 분류한다고 가정해 봅시다. 어떻게 프로그램을 만들어야 할까요?



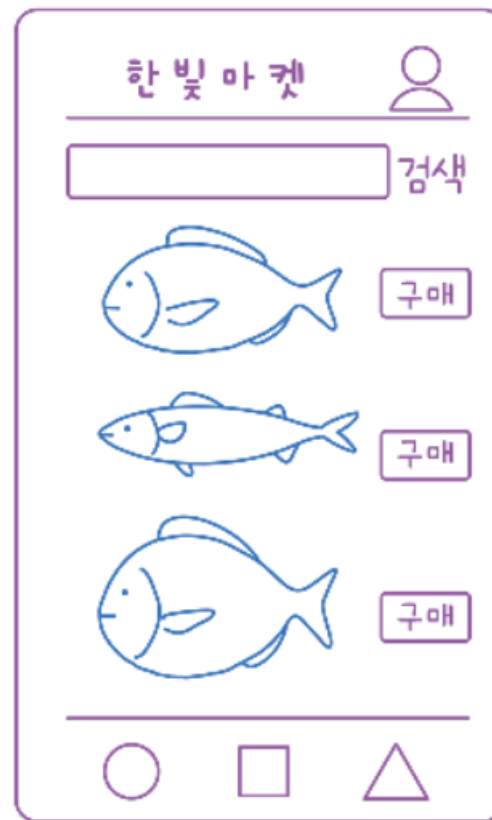
+ 여기서 잠깐

생선 데이터셋의 출처

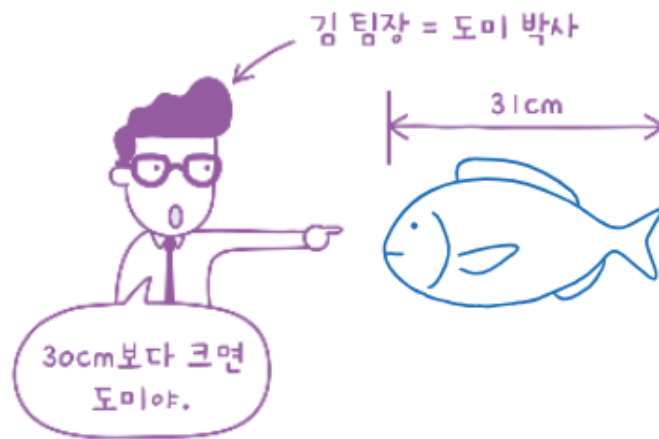
이번에 사용할 생선 데이터는 캐글에 공개된 데이터셋입니다.

- <https://www.kaggle.com/datasets/vipullrathod/fish-market>

캐글(kaggle.com)은 2010년에 설립된 전 세계에서 가장 큰 머신러닝 경연 대회 사이트입니다. 대회 정보뿐만 아니라 많은 데이터와 참고 자료를 제공합니다.



아무래도 생선을 분류하는 일이니 생선의 특징을 알면 쉽게 구분할 수 있을 것 같습니다. 마침 김 팀장이 도미에 대해 잘 안다며 혼공머신에게 생선 길이가 30cm 이상이면 도미라고 알려줬습니다. 그 이야기를 듣고 혼공머신은 다음과 같은 파이썬 프로그램을 만들었습니다.



```
if fish_length >= 30:  
    print("도미")
```

하지만 30cm보다 큰 생선이 무조건 도미라고 말할 수 없습니다. 또 도미의 크기가 모두 같을 리도 없겠죠. 물론 고래와 새우처럼 현격히 차이가 있다면 길이만으로 둘 중 하나를 고르는 프로그램을 만들 수 있을 겁니다. 하지만 한빛 마켓에서 판매하는 생선은 이렇게 절대 바뀌지 않을 기준을 정하기 어렵습니다.

그럼 이 문제를 머신러닝으로 어떻게 해결할 수 있을까요? 보통 프로그램은 ‘누군가 정해진 기준대로 일’을 합니다. 반대로 머신러닝은 누구도 알려주지 않는 기준을 찾아서 일을 합니다. 다시 말해 누가 말해 주지 않아도 머신러닝은 “30~40cm 길이의 생선은 도미이다”라는 기준을 찾는 거죠.

머신러닝은 스스로
기준을 찾아서 일을
합니다.

도미 데이터 준비하기

머신러닝은 어떻게 이런 기준을 스스로 찾을 수 있을까요? 머신러닝은 여러 개의 도미 생선을 보면 스스로 어떤 생선이 도미인지를 구분할 기준을 찾습니다. 그렇다면 도미 생선을 많이 준비해 놔야겠군요!

혼공머신은 문제를 해결하기 위해 물류 창고를 방문했습니다. 물류 창고에서 일하는 방식을 보니 생선을 고를 때 특별한 저울 위에 올리는데, 이 저울은 무게와 길이를 함께 재어 줍니다. 머신러닝만이 아니라 모든 문제는 먼저 간단한 것부터 해결해야 합니다. 먼저 혼공머신은 머신러닝을 사용해 도미와 빙어를 구분하기로 했습니다. 도미와 빙어를 준비해서 저울에 올려놓고 무게와 길이를 재어 보죠.

+ 여기서 잠깐

이진 분류

머신러닝에서 여러 개의 종류(혹은 **클래스(class)**라고 부릅니다) 중 하나를 구별해 내는 문제를 **분류(classification)**라고 부릅니다. 특히 이 장에서처럼 2개의 클래스 중 하나를 고르는 문제를 **이진 분류(binary classification)**라고 합니다. 여기에서 클래스는 파이썬 프로그램의 클래스와는 다릅니다. 혼동하지 마세요.



7일 : ~ 12,000,000

한달 : ~ 53,000,000

출처: <https://hsreplay.net/>

NERFNOW.COM 출처

게임 밸런스 패치방법



길이와 무게를 입력하기 전에 코랩에서 새 노트를 하나 만들어야 합니다. 코랩 메뉴에서 [파일]-[새 노트]를 클릭해서 노트를 생성한 다음에 제목은 'BreamAndSmelt'라고 수정합니다. 그러면 다음과 같은 화면이 보일 겁니다. 이제 그림의 커서가 위치한 곳에 이후 과정을 입력하면 됩니다.



혼공머신은 35마리의 도미를 준비했습니다. 저울로 잰 도미의 길이(cm)와 무게(g)를 파이썬 리스트로 만들면 다음과 같습니다.

note 이 숫자를 손으로 모두 입력하려면 번거롭고 오타를 낼 수 있습니다. http://bit.ly/bream_list에서 복사해서 사용하세요.

```
bream_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7,
31.0, 31.0, 31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5, 33.5,
34.0, 34.0, 34.5, 35.0, 35.0, 35.0, 35.0, 36.0, 36.0, 37.0,
38.5, 38.5, 39.5, 41.0, 41.0]
```

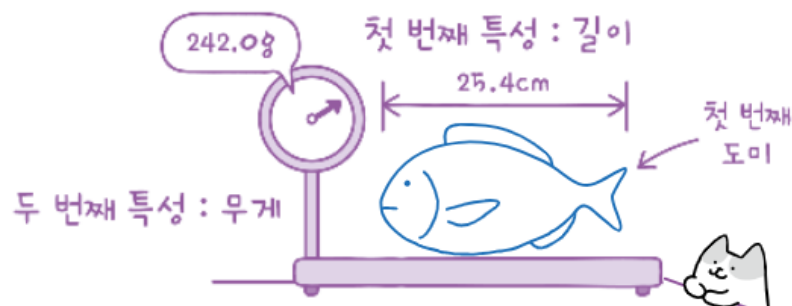
↓
생선의 길이

```
bream_weight = [242.0, 290.0, 340.0, 363.0, 430.0, 450.0, 500.0, 390.0,
450.0, 500.0, 475.0, 500.0, 500.0, 340.0, 600.0, 600.0,
700.0, 700.0, 610.0, 650.0, 575.0, 685.0, 620.0, 680.0,
700.0, 725.0, 720.0, 714.0, 850.0, 1000.0, 920.0, 955.0,
925.0, 975.0, 950.0]
```

↓
생선의 무게

리스트에서 첫 번째 도미의 길이는 25.4cm, 무게는 242.0g이고 두 번째 도미의 길이는 26.3cm, 무게는 290.0g과 같은 식입니다. 각 도미의 특징을 길이와 무게로 표현한 것이죠. 이 책에서는 이런 특징을 **특성** ^{feature}이라고 부르겠습니다.

특성은 데이터의
특징입니다.



두 특성을 숫자로 보는 것보다 그래프로 표현하면 데이터를 잘 이해할 수 있고 앞으로 할 작업에 대한 힌트를 얻을 수도 있습니다. 길이를 x축으로 하고 무게를 y축으로 정하겠습니다. 그다음 각 도미를 이 그래프에 점으로 표시해 보죠. 이런 그래프를 **산점도** scatter plot라고 부릅니다.

산점도는 x, y축으로 이뤄진 좌표계에 두 변수(x, y)의 관계를 표현하는 방법입니다.

파이썬에서 과학계산용 그래프를 그리는 대표적인 패키지는 **맷플롯립** matplotlib입니다. 이 패키지를 임포트하고 산점도를 그리는 `scatter()` 함수를 사용해 보겠습니다. **임포트** import란 따로 만들어둔 파이썬 패키지(클래스와 함수의 묶음)를 사용하기 위해 불러오는 명령입니다.

+ 여기서 잠깐

코랩에서의 패키지와 as

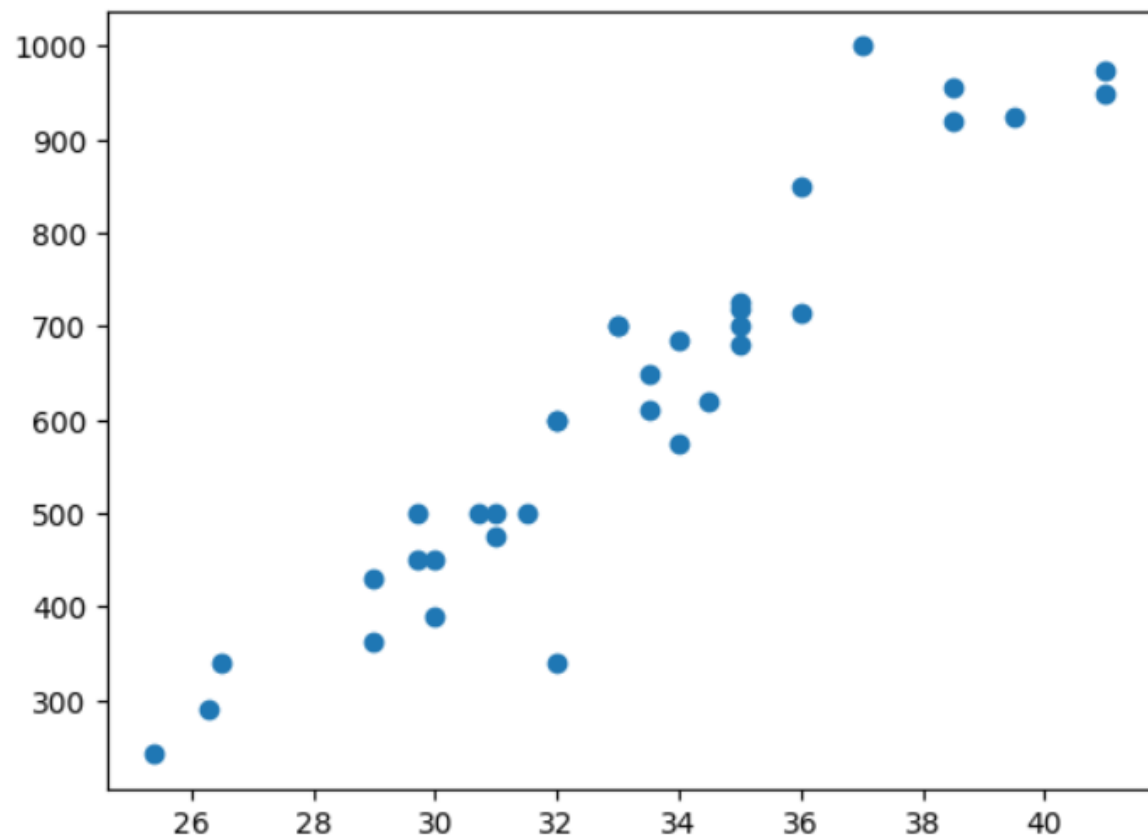
패키지는 기능을 구현한 함수를 특정 기능별로 묶어둔 것입니다. 보통은 이런 패키지를 따로 설치해야 합니다. 하지만 코랩에서는 맷플롯립 같은 패키지를 따로 설치할 필요가 없습니다. 코랩은 널리 사용되는 파이썬 과학 패키지를 미리 준비해 놓았습니다. 이 책에서 사용되는 모든 파이썬 패키지는 코랩에 이미 설치되어 있기 때문에 바로 사용할 수 있습니다. 만약 코랩을 사용하지 않고, 자신의 컴퓨터에서 이 책의 코드를 따라하고 싶다면 <https://tensorflow.blog/install-python>을 참고하세요. 파이썬 프로그래머들은 패키지를 임포트할 때 as 키워드로 패키지 이름을 줄여서 쓰는 것을 좋아합니다. 대표적인 파이썬 패키지들은 이미 널리 사용되는 줄임말이 있습니다. plt도 그중에 하나입니다. 이 책에서는 이런 관용적인 줄임말을 많이 사용합니다. 코드를 읽기 쉽게 만들려면 널리 사용되는 스타일을 따르는 것이 좋습니다.


```
bream_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7,  
30.0, 30.0, 30.7, 31.0, 31.0,  
31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5,  
33.5, 34.0, 34.0, 34.5, 35.0,  
35.0, 35.0, 35.0, 36.0, 36.0, 37.0, 38.5,  
38.5, 39.5, 41.0, 41.0]  
  
bream_weight = [242.0, 290.0, 340.0, 363.0, 430.0, 450.0,  
500.0, 390.0, 450.0, 500.0, 475.0, 500.0,  
500.0, 340.0, 600.0, 600.0, 700.0, 700.0,  
610.0, 650.0, 575.0, 685.0, 620.0, 680.0,  
700.0, 725.0, 720.0, 714.0, 850.0, 1000.0,  
920.0, 955.0, 925.0, 975.0, 950.0]
```

```
import matplotlib.pyplot as plt  
  
plt.scatter(bream_length, bream_weight)  
plt.xlabel('length')  
plt.ylabel('weight')  
plt.show()  
  
%matplotlib inline
```

```
AttributeError                                Traceback (most recent call last)
<ipython-input-2-1f2f8dae3bec> in <cell line: 0>()
     9
    10 plt.scatter(bream_length, bream_weight)
--> 11 plt.xtable('length')
    12 plt.ylable('weight')
    13 plt.show()
```

AttributeError: module 'matplotlib.pyplot' has no attribute 'xtable'

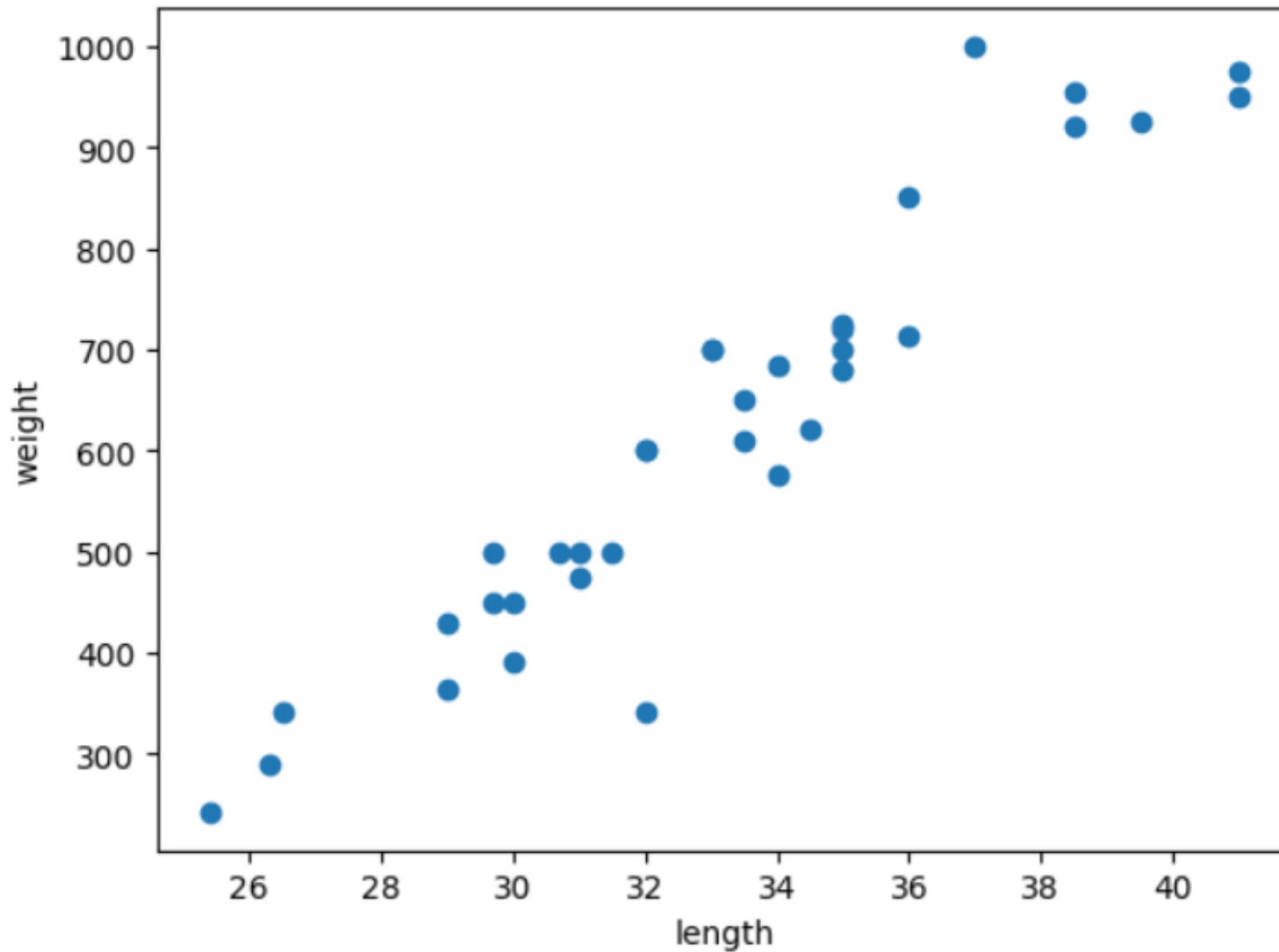



```
bream_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7,  
30.0, 30.0, 30.7, 31.0, 31.0, 31.0, 31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5,  
33.5, 34.0, 34.0, 34.5, 35.0, 35.0, 35.0, 36.0, 36.0, 37.0, 38.5,  
38.5, 39.5, 41.0, 41.0]  
  
bream_weight = [242.0, 290.0, 340.0, 363.0, 430.0, 450.0,  
500.0, 390.0, 450.0, 500.0, 475.0, 500.0, 500.0, 340.0, 600.0, 600.0, 700.0, 700.0,  
610.0, 650.0, 575.0, 685.0, 620.0, 680.0, 700.0, 725.0, 720.0, 714.0, 850.0, 1000.0,  
920.0, 955.0, 925.0, 975.0, 950.0]
```

```
import matplotlib.pyplot as plt  
  
plt.scatter(bream_length, bream_weight)  
plt.xlabel('length')  
plt.ylabel('weight')  
plt.show()
```

$n = 35$ $x = \text{길이}$ $y = \text{무게}$

Linear (선형) 그래프



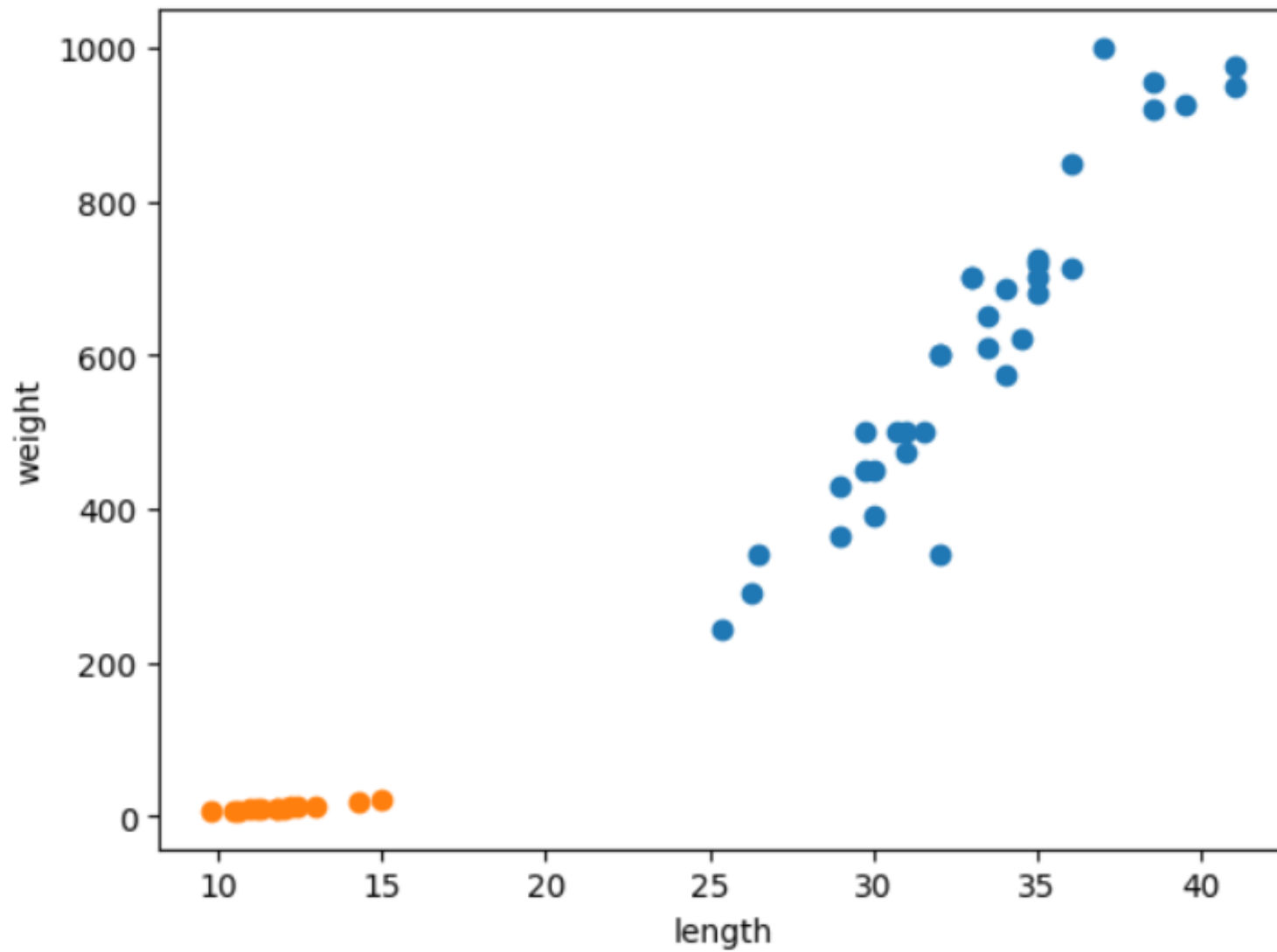
빙어 데이터 준비하기

물류 센터에는 빙어가 많지 않아 혼공머신이 준비한 빙어는 14마리입니다. 앞서와 같이 파이썬 리스트로 만들어 보겠습니다.

note 이 숫자를 손으로 모두 입력하려면 번거롭습니다. http://bit.ly/smelt_list에서 복사해 사용하세요.

손코딩

```
smelt_length = [ 9.8, 10.5, 10.6, 11.0, 11.2, 11.3, 11.8, 11.8, 12.0, 12.2,
                 12.4, 13.0, 14.3, 15.0]
smelt_weight = [ 6.7, 7.5, 7.0, 9.7, 9.8, 8.7, 10.0, 9.9, 9.8, 12.2, 13.4,
                 12.2, 19.7, 19.9]
```



첫 번째 머신러닝 프로그램

여기에서는 가장 간단하고 이해하기 쉬운 k-최근접 이웃 k-Nearest Neighbors 알고리즘을 사용해 도미와 빙어 데이터를 구분해 보겠습니다.

k-최근접 이웃 알고리즘을 사용하기 전에 앞에서 준비했던 도미와 빙어 데이터를 하나의 데이터로 합치겠습니다. 파이썬에서는 다음처럼 두 리스트를 더하면 하나의 리스트로 만들어 줍니다.

손코딩

```
length = bream_length + smelt_length
weight = bream_weight + smelt_weight
```

아주 간단하게 두 리스트를 하나로 합쳤습니다. 이제 length와 weight 리스트는 다음과 같습니다.

도미 35개의 길이
빙어 14개의 길이

└──────────┘
└────────┘

length = [25.4, 26.3, ..., 41.0, 9.8, ..., 15.0]

도미 35개의 무게
빙어 14개의 무게

└──────────┘
└────────┘

weight = [242.0, 290.0, ..., 950.0, 6.7, ..., 19.9]

```
print(length)
[25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0,
 30.0, 30.7, 31.0, 31.0, 31.5, 32.0, 32.0, 32.0,
 33.0, 33.0, 33.5, 33.5, 34.0, 34.0, 34.5, 35.0,
 35.0, 35.0, 35.0, 36.0, 36.0, 37.0, 38.5, 38.5,
 39.5, 41.0, 41.0, 9.8, 10.5, 10.6, 11.0, 11.2,
 11.3, 11.8, 11.8, 12.0, 12.2, 12.4, 13.0, 14.3,
 15.0]
```

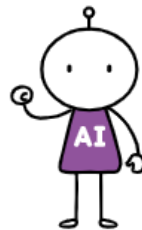
```
print(weight)
[242.0, 290.0, 340.0, 363.0, 430.0, 450.0,
 500.0, 390.0, 450.0, 500.0, 475.0, 500.0,
 500.0, 340.0, 600.0, 600.0, 700.0, 700.0,
 610.0, 650.0, 575.0, 685.0, 620.0, 680.0,
 700.0, 725.0, 720.0, 714.0, 850.0, 1000.0,
 920.0, 955.0, 925.0, 975.0, 950.0, 6.7, 7.5,
 7.0, 9.7, 9.8, 8.7, 10.0, 9.9, 9.8, 12.2, 13.4,
 12.2, 19.7, 19.9]
```

이 책에서 사용하는 머신러닝 패키지는 사이킷런 scikit-learn입니다. 이 패키지를 사용하려면 다음처럼 각 특성의 리스트를 세로 방향으로 늘어뜨린 2차원 리스트를 만들어야 합니다.

데이터 전처리

49개의 생선

	길이	무게
{	[[25.4,	242.0],
	[26.3,	290.0],
	.	.
	.	.
	.	.
	[15.0,	19.9]]



사이킷런은 머신러닝 패키지이며 2차원 리스트가 필요합니다.

이렇게 만드는 가장 쉬운 방법은 파이썬의 zip() 함수와 리스트 내포 list comprehension 구문을 사용하는 것입니다. zip() 함수는 나열된 리스트 각각에서 하나씩 원소를 꺼내 반환합니다. zip() 함수와 리스트 내포 구문을 사용해 length와 weight 리스트를 2차원 리스트로 만들어 보겠습니다.

손코딩

```
fish_data = [[l, w] for l, w in zip(length, weight)]
```

+ 여기서 잠깐**zip() 함수와 for 문**

이 책은 파이썬 입문서가 아니므로 문법 파트가 없습니다. 하지만 파이썬을 잘 모를 독자를 위해 간단하게 설명합니다.

함수란 특정 기능을 실행하는 명령의 코드 모음입니다. 따라서 함수에 따라 실행하는 특정 기능이 있는데 zip() 함수는 나열된 리스트에서 원소를 하나씩 꺼내주는 일을 합니다. 이렇게 하나씩 꺼낸 데이터에 동일한 작업을 계속 반복하는 일을 해 주는 게 바로 for 반복문입니다. 사용법은 이어지는 코드를 보며 익히길 바랍니다.

for 문은 zip() 함수로 length와 weight 리스트에서 원소를 하나씩 꺼내어 l과 w에 할당합니다. 그러면 [l, w]가 하나의 원소로 구성된 리스트가 만들어집니다. 예상대로 fish_data가 만들어졌는지 출력해서 확인해 봅시다.

손코딩

```
print(fish_data)
```

```
[[25.4, 242.0], [26.3, 290.0], [26.5, 340.0], [29.0, 363.0], [29.0, 430.0],
 [29.7, 450.0], [29.7, 500.0], [30.0, 390.0], [30.0, 450.0], [30.7, 500.0],
 [31.0, 475.0], [31.0, 500.0], [31.5, 500.0], [32.0, 340.0], [32.0, 600.0],
 [32.0, 600.0], [33.0, 700.0], [33.0, 700.0], [33.5, 610.0], [33.5, 650.0],
 [34.0, 575.0], [34.0, 685.0], [34.5, 620.0], [35.0, 680.0], [35.0, 700.0],
 [35.0, 725.0], [35.0, 720.0], [36.0, 714.0], [36.0, 850.0], [37.0, 1000.0],
 [38.5, 920.0], [38.5, 955.0], [39.5, 925.0], [41.0, 975.0], [41.0, 950.0],
 [9.8, 6.7], [10.5, 7.5], [10.6, 7.0], [11.0, 9.7], [11.2, 9.8], [11.3, 8.7],
 [11.8, 10.0], [11.8, 9.9], [12.0, 9.8], [12.2, 12.2], [12.4, 13.4],
 [13.0, 12.2], [14.3, 19.7], [15.0, 19.9]]
```

```
print(length)
[25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0,
 30.0, 30.7, 31.0, 31.0, 31.5, 32.0, 32.0, 32.0,
 33.0, 33.0, 33.5, 33.5, 34.0, 34.0, 34.5, 35.0,
 35.0, 35.0, 35.0, 36.0, 36.0, 37.0, 38.5, 38.5,
 39.5, 41.0, 41.0, 9.8, 10.5, 10.6, 11.0, 11.2,
 11.3, 11.8, 11.8, 12.0, 12.2, 12.4, 13.0, 14.3,
 15.0]
```

```
print(weight)
[242.0, 290.0, 340.0, 363.0, 430.0, 450.0,
 500.0, 390.0, 450.0, 500.0, 475.0, 500.0,
 500.0, 340.0, 600.0, 600.0, 700.0, 700.0,
 610.0, 650.0, 575.0, 685.0, 620.0, 680.0,
 700.0, 725.0, 720.0, 714.0, 850.0, 1000.0,
 920.0, 955.0, 925.0, 975.0, 950.0, 6.7, 7.5,
 7.0, 9.7, 9.8, 8.7, 10.0, 9.9, 9.8, 12.2, 13.4,
 12.2, 19.7, 19.9]
```


첫 번째 생선의 길이 25.4cm와 무게 242.0g이 하나의 리스트를 구성하고 이런 리스트가 모여 전체 리스트를 만들었습니다. 이런 리스트를 2차원 리스트 혹은 리스트의 리스트라고 부릅니다.

Train set

생선 49개의 길이와 무게를 모두 준비했습니다. 마지막으로 준비할 데이터는 정답 데이터입니다. 즉 첫 번째 생선은 도미이고, 두 번째 생선도 도미라는 식으로 각각 어떤 생선인지 답을 만드는 것입니다. 왜 이런 작업이 필요할까요?

혼공머신은 머신러닝 알고리즘이 생선의 길이와 무게를 보고 도미와 빙어를 구분하는 규칙을 찾기를 원합니다. 그렇게 하려면 적어도 어떤 생선이 도미인지 빙어인지를 알려 주어야 합니다. 만약 스무개를 하는데 고개마다 답을 알려 주지 않는다면 정답을 맞힐 수 없는 것과 비슷합니다.

머신러닝은 물론이고 컴퓨터 프로그램은 문자를 직접 이해하지 못합니다. 대신 도미와 빙어를 숫자 1과 0으로 표현해 보겠습니다. 예를 들어 첫 번째 생선은 도미이므로 1이고 마지막 생선은 빙어이므로 0이 됩니다.

앞서 도미와 빙어를 순서대로 나열했기 때문에 정답 리스트는 1이 35번 등장하고 0이 14번 등장하면 됩니다. 곱셈 연산자를 사용하면 파이썬 리스트를 간단하게 반복시킬 수 있습니다.

손코딩


```
fish_target = [1] * 35 + [0] * 14  
print(fish_target)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Annotation


note 앞으로 배우게 되겠지만 머신러닝에서 2개를 구분하는 경우 찾으려는 대상을 1로 놓고 그 외에는 0으로 놓습니다. 위의 예는 도미를 찾는 대상으로 정의했기 때문에 도미를 1로 놓고 빙어를 0으로 놓았습니다. 반대로 빙어를 찾는 대상으로 놓고 빙어를 1로 놓아도 됩니다.

이제 사이킷런 패키지에서 k-최근접 이웃 알고리즘을 구현한 클래스인 KNeighborsClassifier를 임포트합니다.

 손코딩

```
from sklearn.neighbors import KNeighborsClassifier
```

+ 여기서 잠깐

from ~ import 구문 

파이썬에서 패키지나 모듈 전체를 임포트하지 않고 특정 클래스만 임포트하려면 from ~ import 구문을 사용합니다. 이렇게 하면 다음과 같이 클래스 이름을 길게 사용하지 않아도 됩니다.

```
import sklearn  
model = sklearn.neighbors.KNeighborsClassifier()
```

손코딩

```
kn = KNeighborsClassifier()
```

이 객체에 fish_data와 fish_target을 전달하여 도미를 찾기 위한 기준을 학습시킵니다. 이런 과정을 머신러닝에서는 훈련 training이라고 부릅니다. 사이킷런에서는 fit() 메서드가 이런 역할을 합니다. 이 메서드에 fish_data와 fish_target을 순서대로 전달해 보겠습니다.

모델에 데이터를 전달하여 규칙을 학습하는 과정을 훈련이라고 합니다.

손코딩

```
kn.fit(fish_data, fish_target)
```


fit() 메서드는 주어진 데이터로 알고리즘을 훈련합니다.

이제 객체(또는 모델) kn이 얼마나 잘 훈련되었는지 평가해 보겠습니다. 사이킷런에서 모델을 평가하는 메서드는 score() 메서드입니다. 이 메서드는 0에서 1 사이의 값을 반환합니다. 1은 모든 데이터를 정확히 맞췄다는 것을 나타냅니다. 예를 들어 0.5라면 절반만 맞췄다는 의미죠.


+ 여기서 잠깐

머신러닝에서의 모델

머신러닝 알고리즘을 구현한 프로그램을 **모델(model)**이라고 부릅니다. 또는 프로그램이 아니더라도 알고리즘을 (수식 등으로) 구체화하여 표현한 것을 모델이라고 부릅니다. 예를 들어 “스팸 메일을 걸러내기 위해 k-최근접 이웃 모델을 사용해 봅시다”라고 말할 수 있습니다.

손코딩

```
kn.score(fish_data, fish_target)
```

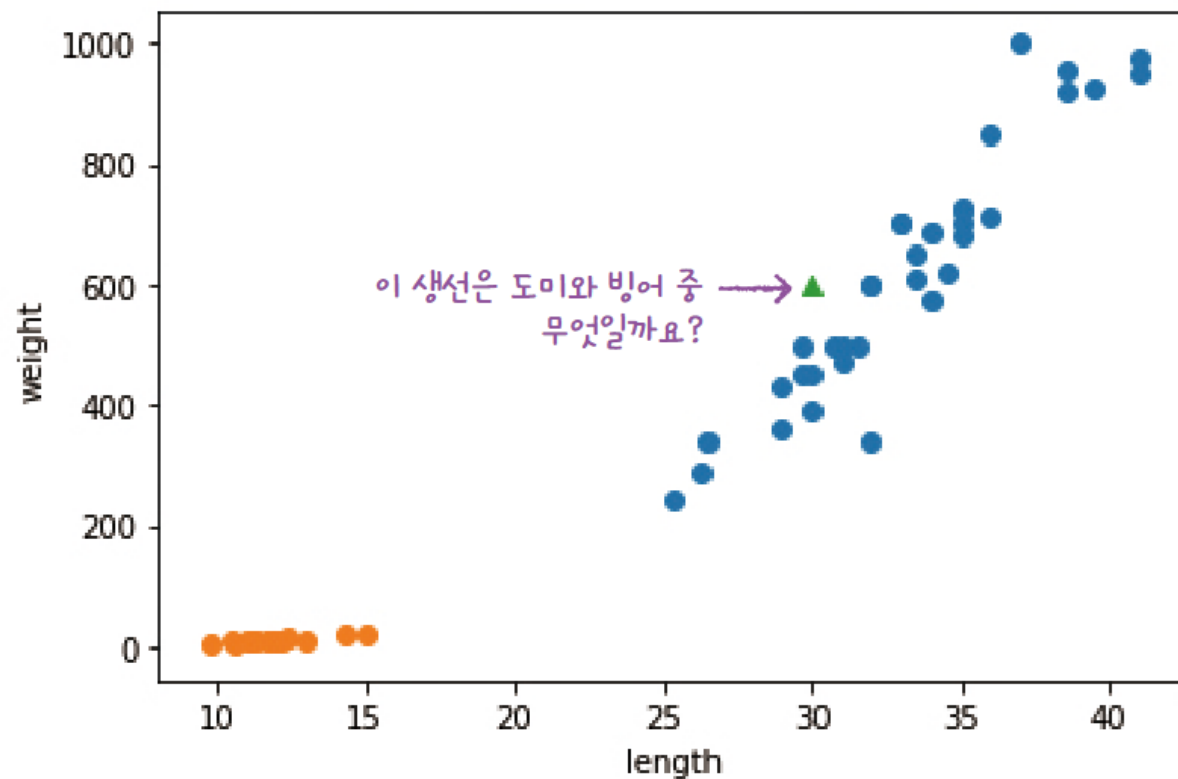
 1.0

와우! 1.0이네요. 모든 fish_data의 답을 정확히 맞혔습니다. 이 값을 정확도 accuracy라고 부릅니다. 즉 이 모델은 정확도가 100%이며 도미와 빙어를 완벽하게 분류했습니다. 성공이네요. 혼공머신은 이제 퇴근해도 될 것 같습니다!

k-최근접 이웃 알고리즘

앞에서 첫 번째 머신러닝 프로그램을 성공적으로 만들었습니다. 여기에서 사용한 알고리즘은 k-최근접 이웃입니다. 이 알고리즘에 대해 조금 더 자세히 알아보도록 하겠습니다. k-최근접 이웃 알고리즘은 매우 간단합니다. 어떤 데이터에 대한 답을 구할 때 주위의 다른 데이터를 보고 다수를 차지하는 것을 정답으로 사용합니다. 마치 근목자흑과 같이 주위의 데이터로 현재 데이터를 판단하는 거죠.

예를 들어 다음 그림에 삼각형으로 표시된 새로운 데이터가 있다고 가정해 보죠. 이 삼각형은 도미와 빙어 중 어디에 속할까요?



아마도 여러분은 직관적으로 이 삼각형은 도미라고 판단할 것입니다. 왜냐하면 삼각형 주변에 다른 도미 데이터가 많기 때문이죠. k -최근접 이웃 알고리즘도 마찬가지입니다. 이 삼각형 주위에 도미 데이터가 많으므로 삼각형을 도미라고 판단할 것입니다. 실제로 그런지 한번 확인해 보죠.

손코딩

```
kn.predict([[30, 600]])
```



```
array([1])
```


★ predict() 메서드는 새로운 데이터의 정답을 예측합니다. 이 메서드도 앞서 fit() 메서드와 마찬가지로 리스트의 리스트를 전달해야 합니다. 그래서 삼각형 포인트를 리스트로 2번 감쌌습니다. 반환되는 값은 1입니다. 우리는 앞서 도미는 1, 빙어는 0으로 가정했습니다. 즉 삼각형은 도미입니다. 예상과 같네요.

note 반환되는 값에 나타난 array()는 잠시 잊어도 좋습니다. 나중에 자세히 설명하겠습니다.

이렇게 생각하면 k -최근접 이웃 알고리즘을 위해 준비해야 할 일은 데이터를 모두 가지고 있는 게 전부입니다. 새로운 데이터에 대해 예측할 때는 가장 가까운 직선거리에 어떤 데이터가 있는지를 살펴보기만 하면 됩니다. 단점은 k -최근접 이웃 알고리즘의 이런 특징 때문에 데이터가 아주 많은 경우 사용하기 어렵습니다. 데이터가 크기 때문에 메모리가 많이 필요하고 직선거리를 계산하는 데도 많은 시간이 필요합니다.

좋습니다. 실제로 k-최근접 이웃 알고리즘은 무언가 훈련되는 게 없는 셈이네요. fit() 메서드에 전달한 데이터를 모두 저장하고 있다가 새로운 데이터가 등장하면 가장 가까운 데이터를 참고하여 도미인지 빙어인지를 구분합니다.

그럼 가까운 몇 개의 데이터를 참고할까요? 이는 정하기 나름입니다. KNeighborsClassifier 클래스의 기본값은 5입니다. 이 기준은 n_neighbors 매개변수로 바꿀 수 있습니다. 예를 들어 다음과 같이 하면 어떤 결과가 나올까요?

손코딩

```
kn49 = KNeighborsClassifier(n_neighbors=49) # 참고 데이터를 49개로 한 kn49 모델
```

가장 가까운 데이터 49개를 사용하는 k-최근접 이웃 모델에 fish_data를 적용하면 fish_data에 있는 모든 생선을 사용하여 예측하게 됩니다. 다시 말하면 fish_data의 데이터 49개 중에 도미가 35개로 다수를 차지하므로 어떤 데이터를 넣어도 무조건 도미로 예측할 것입니다.


 손코딩

```
kn49.fit(fish_data, fish_target)
kn49.score(fish_data, fish_target)
```



0.7142857142857143

fish_data에 있는 생선 중에 도미가 35개이고 빙어가 14개입니다. kn49 모델은 도미만 올바르게 맞히기 때문에 다음과 같이 정확도를 계산하면 score() 메서드와 같은 값을 얻을 수 있습니다.

 손코딩

```
print(35/49)
```



```
0.7142857142857143
```

확실히 n_neighbors 매개변수를 49로 두는 것은 좋지 않네요. 기본값을 5로 하여 도미를 완벽하게 분류한 모델을 사용하겠습니다.

혼공머신은 김 팀장에게 이 기쁜 소식을 전달했습니다.

도미와 빙어 분류

문제해결 과정

혼공머신은 도미와 빙어를 구분하기 위해 첫 번째 머신러닝 프로그램을 만들었습니다. 먼저 도미 35마리와 빙어 14마리의 길이와 무게를 측정해서 파이썬 리스트로 만듭니다. 그다음 도미와 빙어 데이터를 합쳐 리스트의 리스트로 데이터를 준비했습니다.

혼공머신이 사용한 첫 번째 머신러닝 알고리즘은 k-최근접 이웃 알고리즘입니다. 사이킷런의 k-최근접 이웃 알고리즘은 주변에서 가장 가까운 5개의 데이터를 보고 다수결의 원칙에 따라 데이터를 예측합니다. 이 모델은 혼공머신이 준비한 도미와 빙어 데이터를 모두 완벽하게 맞혔습니다.

도미와 빙어를 분류하는 문제를 풀면서 KNeighborsClassifier 클래스의 `fit()`, `score()`, `predict()` 메서드를 사용해 보았습니다. 끝으로 k-최근접 이웃 알고리즘의 특징을 알아보았습니다.

▶ 키워드로 끝내는 핵심 포인트

- **특성**은 데이터를 표현하는 하나의 성질입니다. 이 절에서 생선 데이터 각각을 길이와 무게 특성으로 나타냈습니다.
- 머신러닝 알고리즘이 데이터에서 규칙을 찾는 과정을 **훈련**이라고 합니다. 사이킷런에서는 `fit()` 메서드가 하는 역할입니다.
- **k-최근접 이웃 알고리즘**은 가장 간단한 머신러닝 알고리즘 중 하나입니다. 사실 어떤 규칙을 찾기보다는 전체 데이터를 메모리에 가지고 있는 것이 전부입니다.
- 머신러닝 프로그램에서는 알고리즘이 구현된 객체를 **모델**이라고 부릅니다. 종종 알고리즘 자체를 모델이라고 부르기도 합니다.
- **정확도**는 정확한 답을 몇 개 맞혔는지를 백분율로 나타낸 값입니다. 사이킷런에서는 0~1 사이의 값으로 출력됩니다.

$$\text{정확도} = (\text{정확히 맞힌 개수}) / (\text{전체 데이터 개수})$$

▶ 핵심 패키지와 함수

matplotlib

- **scatter()**는 산점도를 그리는 맷플롯립 함수입니다. 처음 2개의 매개변수로 x축 값과 y축 값을 전달합니다. 이 값은 파이썬 리스트 또는 넘파이(다음 절에서 소개합니다) 배열입니다.

c 매개변수로 색깔을 지정합니다. RGB를 16진수(예를 들면 '#1f77b4')로 지정하거나 색깔 코드 'b'(파랑), 'g'(초록), 'r'(빨강), 'c'(시안), 'm'(마젠타), 'y'(노랑), 'k'(검정), 'w'(흰색) 중 하나를 지정합니다. 지정하지 않을 경우 10개의 기본 색깔을 사용해 그래프를 그립니다. 기본 색깔은 https://bit.ly/matplotlib_prop_cycle을 참고하세요.

marker 매개변수로 마커 스타일을 지정합니다. marker의 기본값은 o(circle, 원)입니다. 지정할 수 있는 마커 종류는 https://bit.ly/matplotlib_marker를 참고하세요.

scikit-learn

- **KNeighborsClassifier()**는 k-최근접 이웃 분류 모델을 만드는 사이킷런 클래스입니다. `n_neighbors` 매개변수로 이웃의 개수를 지정합니다. 기본값은 5입니다.
`p` 매개변수로 거리를 재는 방법을 지정합니다. 1일 경우 맨해튼 거리(https://bit.ly/man_distance)를 사용하고, 2일 경우 유클리디안 거리(https://bit.ly/euc_distance)를 사용합니다. 기본값은 2입니다.
`n_jobs` 매개변수로 사용할 CPU 코어를 지정할 수 있습니다. -1로 설정하면 모든 CPU 코어를 사용합니다. 이웃 간의 거리 계산 속도를 높일 수 있지만 `fit()` 메서드에는 영향이 없습니다. 기본값은 1입니다.
- **fit()**은 사이킷런 모델을 훈련할 때 사용하는 메서드입니다. 처음 두 매개변수로 훈련에 사용할 특성과 정답 데이터를 전달합니다.
- **predict()**는 사이킷런 모델을 훈련하고 예측할 때 사용하는 메서드입니다. 특성 데이터 하나만 매개변수로 받습니다.
- **score()**는 훈련된 사이킷런 모델의 성능을 측정합니다. 처음 두 매개변수로 특성과 정답 데이터를 전달합니다. 이 메서드는 먼저 `predict()` 메서드로 예측을 수행한 다음 분류 모델일 경우 정답과 비교하여 올바르게 예측한 개수의 비율을 반환합니다.

▶ 확인 문제

1. 데이터를 표현하는 하나의 성질로써, 예를 들어 국가 데이터의 경우 인구 수, GDP, 면적 등이 하나의 국가를 나타냅니다. 머신러닝에서 이런 성질을 무엇이라고 부르나요?

- ① 특성
- ② 특질
- ③ 개성
- ④ 요소

2. 가장 가까운 이웃을 참고하여 정답을 예측하는 알고리즘이 구현된 사이킷런 클래스는 무엇인가요?

- ① SGDClassifier
- ② LinearRegression
- ③ RandomForestClassifier
- ④ KNeighborsClassifier

3. 사이킷런 모델을 훈련할 때 사용하는 메서드는 어떤 것인가요?

- ① predict()
- ② fit()
- ③ score()
- ④ transform()

4. 본문^{56쪽}에서 `n_neighbors`를 49로 설정했을 때 점수가 1.0보다 작았습니다. 즉 정확도가 100%가 아닙니다. 그럼 `n_neighbors`의 기본값인 5부터 49까지 바꾸어 가며 점수가 1.0 아래로 내려가기 시작하는 이웃의 개수를 찾아보세요. 이 문제를 위해 `KNeighborsClassifier` 클래스 객체를 매번 다시 만들 필요는 없습니다. 심지어 `fit()` 메서드로 훈련을 다시 할 필요도 없습니다. `k-최근접 이웃` 알고리즘의 훈련은 데이터를 저장하는 것이 전부이기 때문입니다. `KNeighborsClassifier` 클래스의 이웃 개수는 모델 객체의 `n_neighbors` 속성으로 바꿀 수 있습니다. 이웃 개수를 바꾼 후 `score()` 메서드로 다시 계산하기만 하면 됩니다.

```
kn = KNeighborsClassifier()
kn.fit(fish_data, fish_target)

for n in range(5, 50):
    # k-최근접 이웃 개수 설정
    kn.n_neighbors =  # 이 라인을 완성해 보세요
    # 점수 계산
    score = kn.score(, ) # 이 라인을 완성해 보세요
    # 100% 정확도에 미치지 못하는 이웃 개수 출력
    if score < 1:
        print(n, score)
        break
```

축하합니다! 여러분은 고생 끝에 수원대학교 데이터과학부의 조교수로 임용 받았습니다!

데이터과학부는 한 학년에 110명이나 된다고 합니다!

110명은 서로 다른 성격과 특성을 가지고 있어서, 이 학생들을 어떻게 분류해서 수업을 진행할지 막막합니다.

이제 여러분은 3학년과 4학년, 총 220명을 맡아서 학생들의 학습 성취도를 학생의 성향에 맞는 방법으로 찾아서 분류하려고 합니다.

학습 성취도는 보통 시험과 프로젝트로 나뉘는다고 합니다. 어떤 학생은 시험을 잘 보고 어떤 학생은 프로젝트를 잘 하는 거죠. 물론 둘 사이에는 어느정도 연관관계가 있어서 시험은 0점인데 프로젝트는 100점을 받는 학생은 없다고 가정합시다. 여러분은 최대한 학생들에게 유리하게 수업을 나누어 진행하려고 합니다.

그리고보니 수업을 진행하면서 알게 된 KNeighborsClassifier를 사용해보고 싶습니다.

주어진 데이터 세트를 사용하여 아래 그래프를 그리세요.

위 데이터 세트에서 최근접 이웃 데이터 3개를 사용하여서 모델을 훈련하고 예측 스코어를 구해보세요.
훈련한 모델을 사용하여 아래 학생의 학습 향상을 위해서는 어떤 학습 성취 방법을 사용해야 하는지
예측해보세요.

- 학생 1: 중간고사 60점, 프로젝트 90점
- 학생 2: 중간고사 90점, 프로젝트 60점
- 학생 3: 중간고사 80점, 프로젝트 81점

위 데이터 세트에서 최근접 이웃 데이터 30개를 사용하여서 모델을 훈련하고 예측 스코어를 구해보세요.
훈련한 모델을 사용하여 아래 학생의 학습 향상을 위해서는 어떤 학습 성취 방법을 사용해야 하는지
예측해보세요.

- 학생 1: 중간고사 60점, 프로젝트 90점
- 학생 2: 중간고사 90점, 프로젝트 60점
- 학생 3: 중간고사 80점, 프로젝트 81점

학습문제: https://github.com/mario2437/ML_lecture_1/blob/main/You%20are%20the%20professor

