# 데이터 과학

## L03.1: Linear Regression Practice

Kookmin University

# Contents

- **Pytorch Basic**
- Linear Regression with Pytorch
- Linear Regression with Scikit-learn

# Pytorch 사용하기

Python에서 Pytorch를 사용하려면, ’pytorch를 사용하겠다’고 선언 해야함

```
import torch
```

# 텐서 Tensor

**텐서**: 파이토치에서 사용하는 가장 기본적인 자료구조

- 다양한 수식 계산을 지원
- 수학의 스칼라, 벡터, 행렬 등을 일반화한 개념.
  - 스칼라 = 0 차원rank 텐서. 예) 1
  - 벡터 = 1 차원 텐서. 예) [1,2]
  - 행렬 = 2 차원 텐서. 예) [[1,2],[3,4]]
  - 3 차원 텐서 예) [[[1,2], [3,4]], [[1,7], [5,4]]]

# 텐서 Tensor

- 텐서 만들기

```
x = torch.tensor([[1,2,3], [4,5,6], [7,8,9]])
y = torch.FloatTensor([[1,2,3], [4,5,6], [7,8,9]])
print("x =", x)
print("y =", y)
```

$$x = \begin{bmatrix} 1, 2, 3 \\ 4, 5, 6 \\ 7, 8, 9 \end{bmatrix} \; y = \begin{bmatrix} 1.0, 2.0, 3.0 \\ 4.0, 5.0, 6.0 \\ 7.0, 8.0, 9.0 \end{bmatrix}$$

# 텐서 Tensor

- 크기와 랭크(차원) 확인하기

```
print("Size:", x.size())
print("Shape:", x.shape)
print("차원(랭크):", x.ndimension())
```

# 텐서 Tensor

- **텐서 모양 바꾸기:**
  - unsqueeze(x, i): tensor x에 i 번째 차원 추가

```
x0 = torch.unsqueeze(x, 0)
x1 = torch.unsqueeze(x, 1)
x2 = torch.unsqueeze(x, 2)
print("x0.shape:", x0.shape)
print("x1.shape:", x1.shape)
print("x2.shape:", x2.shape)
print("x0 =", x0)
print("x1 =", x1)
print("x2 =", x2)
```

# 텐서 Tensor

- **텐서 모양 바꾸기:**
  - squeeze(x): 텐서 x에서 크기가 1인 차원 제거

```
x3 = torch.squeeze(torch.squeeze(x0))
print("x3 =", x3)
print("x3.shape =", x3.shape)
```

# 텐서 Tensor

- **텐서 모양 바꾸기:**
  - x.view([shape]): x를 [shape]의 모양으로 변환

```
x4 = x.view(9)
x5 = x.view(1,3,3)
print("x4 =", x4)
print("x5 =", x5)
```

# 행렬 연산

- xw + b

```
x = torch.FloatTensor([[1,2], [3,4], [5,6]])
w = torch.randn(1,2, dtype=torch.float)
b = torch.randn(3,1, dtype=torch.float)

result = torch.mm(x, torch.t(w)) + b
print(result)
```

# Autograd: 기울기 계산

- requires_grad 값이 설정된 tensor에 대해서, pytorch는 자동으로 기울기를 계산해 줌

```
w = torch.tensor(1.0, requires_grad=True)
a = w*3
l = a**2
l.backward()
print('l을 w로 미분한 값은', w.grad)
```

$$l = a^2 = (3w)^2 = 9w^2$$

# Contents

- Pytorch Basic
- **Linear Regression with Pytorch**
- Linear Regression with Scikit-learn

# Linear Regression

- 학습 데이터 생성

```
import torch


x_train = torch.FloatTensor([[1,2], [3,2], [3,7], [1,1], [1,0]])
y_train = torch.FloatTensor([[4], [8], [23], [1], [-2]])
```

$$y = 2x_1 + 3x_2 - 4$$

# Linear Regression

- W, b 초기화
- Learning Rate 설정

```
W = torch.rand(2,1)
b = torch.rand(1,1)


lr = 0.01
```

# Linear Regression

- 반복횟수 설정
- W와 b의 requires_grad를 True로 설정

```
for epoch in range(3001):
  W.requires_grad_(True)
  b.requires_grad_(True)
```

# Linear Regression

- Hypothesis, cost 설정

```
hypothesis = torch.mm(x_train, W) + b
cost = torch.mean((hypothesis - y_train) ** 2)
```

# Linear Regression

- 경사 계산
- W, b 업데이트

```
cost.backward()
with torch.no_grad() as grd:
  W = W - lr * W.grad
  b = b - lr * b.grad
```

# Linear Regression

- 학습이 잘 되는지 확인하기위한 내용 출력

```
if epoch % 100 == 0:
  print( 'epoch: {}, cost: {:.6f}, W: {}, b: {}'.format(epoch,
                                        cost.item(), W.squeeze(), b))
```

# Linear Regression

- 학습 결과 확인

$$y = 2x_1 + 3x_2 - 4$$

```
epoch: 0, cost: 122.800003, W: tensor([0.3840, 0.7440]), b: tensor([[0.1360]])
epoch: 100, cost: 1.533758, W: tensor([0.6226, 3.1603]), b: tensor([[-1.3651]])
epoch: 200, cost: 0.758241, W: tensor([0.9143, 3.1864]), b: tensor([[-2.1790]])
epoch: 300, cost: 0.389289, W: tensor([1.2106, 3.1417]), b: tensor([[-2.7023]])
epoch: 400, cost: 0.200058, W: tensor([1.4328, 3.1025]), b: tensor([[-3.0705]])
epoch: 500, cost: 0.102813, W: tensor([1.5932, 3.0736]), b: tensor([[-3.3338]])
epoch: 600, cost: 0.052837, W: tensor([1.7084, 3.0528]), b: tensor([[-3.5224]])
epoch: 700, cost: 0.027154, W: tensor([1.7909, 3.0378]), b: tensor([[-3.6576]])
epoch: 800, cost: 0.013955, W: tensor([1.8501, 3.0271]), b: tensor([[-3.7546]])
epoch: 900, cost: 0.007172, W: tensor([1.8926, 3.0194]), b: tensor([[-3.8241]])
epoch: 1000, cost: 0.003686, W: tensor([1.9230, 3.0139]), b: tensor([[-3.8739]])
epoch: 1100, cost: 0.001894, W: tensor([1.9448, 3.0100]), b: tensor([[-3.9096]])
epoch: 1200, cost: 0.000973, W: tensor([1.9604, 3.0072]), b: tensor([[-3.9352]])
epoch: 1300, cost: 0.000500, W: tensor([1.9716, 3.0051]), b: tensor([[-3.9535]])
epoch: 1400, cost: 0.000257, W: tensor([1.9797, 3.0037]), b: tensor([[-3.9667]])
epoch: 1500, cost: 0.000132, W: tensor([1.9854, 3.0026]), b: tensor([[-3.9761]])
epoch: 1600, cost: 0.000068, W: tensor([1.9895, 3.0019]), b: tensor([[-3.9829]])
epoch: 1700, cost: 0.000035, W: tensor([1.9925, 3.0014]), b: tensor([[-3.9877]])
epoch: 1800, cost: 0.000018, W: tensor([1.9946, 3.0010]), b: tensor([[-3.9912]])
epoch: 1900, cost: 0.000009, W: tensor([1.9961, 3.0007]), b: tensor([[-3.9937]])
epoch: 2000, cost: 0.000005, W: tensor([1.9972, 3.0005]), b: tensor([[-3.9955]])
```

# Linear Regression

- x = [5,10]일 때, y의 값은 몇일까?

```
x_test = torch.FloatTensor([[5,10]])
test_result = torch.mm(x_test, W) + b
print(test_result.item())
```

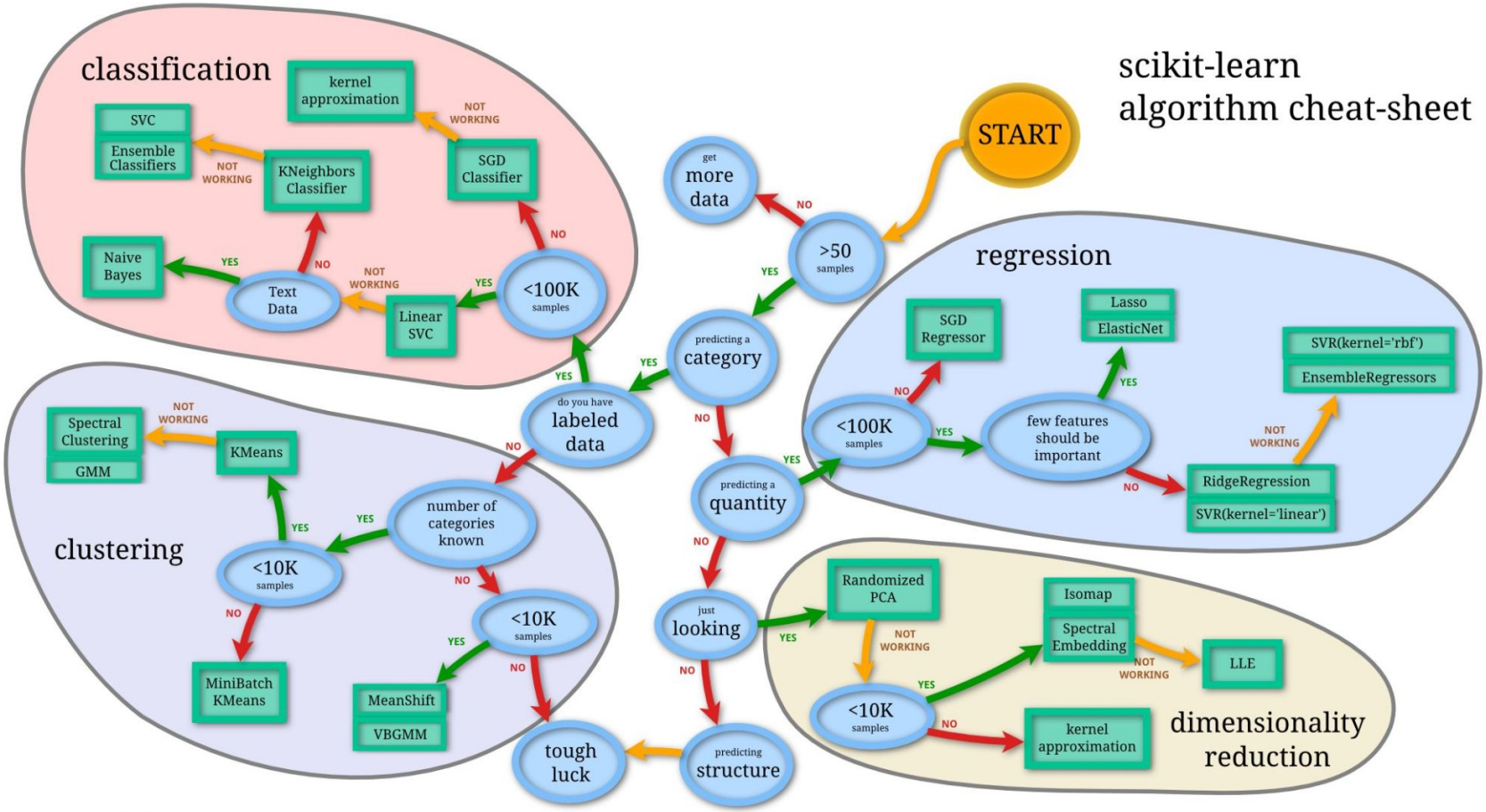
# Contents

- Pytorch Basic
- Linear Regression with Pytorch
- **Linear Regression with Scikit-learn**

# Scikit-learn

## Python의 대표 기계학습 라이브러리

Linear regression을 포함하여 다양한 기계학습 방법을 손쉽게 사용 가능



# Linear Regression in Sklearn

- **모델 생성 및 학습**

```
from sklearn.linear_model import LinearRegression

x = [[1,2], [3,2], [3,7], [1,1], [1,0]]
y = [[4], [8], [23], [1], [-2]]

lr = LinearRegression() # 모델 생성
lr.fit(x, y) # 학습 (피팅)
```

# Linear Regression in Sklearn

- **학습 결과 (Parameter) 확인**
  - coef_: 기울기 값 (W에 해당)
  - intercept_: y절편 값 (b에 해당)

```
print(lr.coef_, lr.intercept_)
```

출력 결과)

```
[[2. 3.]] [-4.]
```

# Linear Regression in Sklearn

- y값 예측하기

```
print(lr.predict([[5,10]]))
```

출력 결과)

```
[[36.]]
```

# Questions?