

# DAY002

## 1. 데이터의 종류 - 스칼라, 벡터, 행렬, 텐서

Scalar	Vector	Matrix	Tensor
1	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$

**Scalar** : 숫자 하나

**Vector** : 숫자들의 배열(스칼라의 집합)

**Matrix** : 2차원 이상의 배열(벡터의 집합)

**Tensor** : 3차원 이상의 배열(행렬의 집합)

아래는 행렬을 코드로 표현한 것이다. 몇 행 몇 열인지 형태를 파악하려면 가장 작은 단위의 개수부터 살펴보면 된다.

```
[1,2,3]#----- (3, )
[[1,2,3], [1,2,3]]#----- (2,3)
[[[1,2,3], [1,2,3]]]#----- (1,2,3)
[[1,2],[3,4]], [[1,2],[3,4]]#----- (2,2,2)
[[1,2,3,4,5],[1,2,3,4,5],[1,2,3,4,5]]#----- (3,5)
```

**[1,2,3] ----- (3, )**

스칼라 3개로 이루어진 3차원 벡터이다. (배열의 차원과 다르다.)

**[[1,2,3], [1,2,3]] ----- (2,3) 2행 3열**

위의 벡터에서 가장 작은 단위는 스칼라의 개수가 3개이고 벡터는 2개이므로 (2,3)이 된다.

## [[[1,2,3], [1,2,3]]] ----- (1,2,3) 1텐서 2행 3열

가장 작은 단위는 스칼라의 개수가 3개이고 벡터는 2개이고 텐서는 1개이므로 (1,2,3)이 된다.

## [[1,2],[3,4]], [[1,2],[3,4]] ----- (2,2,2) 2텐서 2행 2열

가장 작은 단위는 스칼라의 개수가 2개이고 벡터는 2개이고 텐서는 2개이므로 (2,2,2)가 된다.

## [[1,2,3,4,5],[1,2,3,4,5],[1,2,3,4,5]] ----- (3,5) 3행 5열

가장 작은 단위는 스칼라의 개수가 5개이고 벡터는 3개이므로 (3,5)가 된다.

벡터, 행렬, 텐서에서 대해 좀 더 자세히 알아보고 싶다면 아래 링크를 확인하면 된다.

- [Vector 의미 & 연산](#)
- 

## 2. 데이터를 훈련(train), 검증(validation), 평가(test)로 나누는 이유

머신러닝이나 딥러닝에서 모델에 필요한 데이터는 **훈련(train)** / **검증(validation)** / **평가(test)**로 나뉜다.

훈련(train)과 검증(Validation) 데이터 셋은 Training 과정에서 사용되고 실험(test) 데이터는 Test과정에 사용된다.

### 2-1 데이터별 용도

- **훈련(train)데이터** - 알고리즘이 학습할 데이터
- **검증(Validation) 데이터** - Train data로 모델 학습 후 Validation data를 통해 예측/분류 등 학습의 정확도를 계산
- **평가(test) 데이터** - 최종 학습 후 Train data과 Validation data를 제외한 데이터로 모델을 테스트

### 2-2 데이터를 훈련(train), 검증(validation), 평가(test)로 나누는 이유

2-1에서 언급한 데이터별 용도처럼 인공지능이 학습(Train)을 하고 그 학습을 올바르게 하고 있는지 검증(Validation)이 필요하며 그 학습을 마친 후 새로운 데이터가 들어왔을 때 그것 얼마나 정확하게 맞추는지 평가(Test)를 해야 제대로 된 모델인지 알 수 있기 때문이다.

만약 데이터를 나누지 않고 그대로 학습을 시킨다면 과적합(Overfitting)이 되어 다른 데이터가 들어왔을 때 모델은 전혀 예측하지 못할 것이다.

이 학습 데이터 세트를 사용하여 모델을 학습시키고 나면 이후에는 **검증 세트(Validation Set)**를 통해 **모델의 예측/분류 정확도를** 계산할 수 있다. 사실 모든 검증 세트에 대한 **실제 레이블, 즉 정답을 알고 있지만 그렇지 않은 척** 하는 셈이다. 그래서 새로운 데이터인 것처럼 분류/예측 모델에 입력해준다. 실제로 학습 시킬 때 이 데이터들을 배제했기 때문에 가능하다. 예측/분류된 값을 받아서 실제로 갖고 있던 답과 비교하기만 하면 결국 **정확도(Accuracy)**를 알 수 있는 거다.

<http://hleecaster.com/ml-training-validation-test-set/>

```
In [5]: history = model.fit(X_train, y_train, batch_size=64, epochs=100, validation_split=0.15, callbacks=[checkpoint, early_stopping])

Train on 18747 samples, validate on 3309 samples
Epoch 1/100
18747/18747 [=====] - 9s 480us/step - loss: 0.6535 - acc: 0.5976 - val_loss: 0.5828 - val_acc: 0.6878

Epoch 00001: val_loss improved from inf to 0.58284, saving model to ./model/dog_cat_classify.model
Epoch 2/100
18747/18747 [=====] - 5s 258us/step - loss: 0.5458 - acc: 0.7248 - val_loss: 0.4519 - val_acc: 0.7875

Epoch 00002: val_loss improved from 0.58284 to 0.45193, saving model to ./model/dog_cat_classify.model
Epoch 3/100
18747/18747 [=====] - 5s 257us/step - loss: 0.4688 - acc: 0.7803 - val_loss: 0.4365 - val_acc: 0.8005

Epoch 00003: val_loss improved from 0.45193 to 0.43645, saving model to ./model/dog_cat_classify.model
Epoch 4/100
18747/18747 [=====] - 5s 256us/step - loss: 0.4099 - acc: 0.8158 - val_loss: 0.3824 - val_acc: 0.8290

Epoch 00004: val_loss improved from 0.43645 to 0.38235, saving model to ./model/dog_cat_classify.model
Epoch 5/100
18747/18747 [=====] - 5s 253us/step - loss: 0.3696 - acc: 0.8354 - val_loss: 0.3514 - val_acc: 0.8471
```

위 이미지를 보자. epoch가 증가할수록 val\_loss(validation\_loss)가 감소하고 val\_acc(validation accuracy)가 증가되고 있는 것을 알 수 있다.

<https://lsjsj92.tistory.com/545>

## 3. 데이터를 분리하는 방법

train\_test\_split() 메소드를 사용하여 데이터를 분할할 수 있다.

```
from sklearn.model_selection import train_test_split
train_test_split(arrays, test_size, train_size, random_state, shuffle,
stratify)
```

### 3-1 Parameter

**arrays** : 분할시킬 데이터를 입력 (*Python list, Numpy array, Pandas dataframe 등..*)

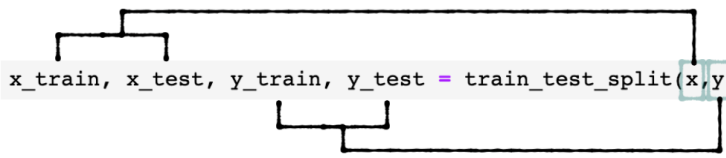
**test\_size** : 테스트 데이터셋의 비율(float)이나 갯수(int) (*default = 0.25*)

**train\_size** : 학습 데이터셋의 비율(float)이나 갯수(int) (*default = test\_size의 나머지*)

**random\_state** : 데이터 분할시 셔플이 이루어지는데 이를 위한 시드값 (*int나 RandomState로 입력*)

**shuffle** : 셔플여부설정 (*default = True*)

**stratify** : 지정한 Data의 비율을 유지한다. 예를 들어, Label Set인 Y가 25%의 0과 75%의 1로 이루어진 Binary Set일 때, stratify=Y로 설정하면 나누어진 데이터셋들도 0과 1을 각각 25%, 75%로 유지한 채 분할된다.



```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, shuffle = False)
```

```
import numpy as np
from sklearn.model_selection import train_test_split

X = [[0,1],[2,3],[4,5],[6,7],[8,9]]
Y = [0,1,2,3,4]

# 데이터(x)만 넣었을 경우
X_train, X_test = train_test_split(X, test_size=0.2, random_state=123)
# X_train : [[0,1],[6,7],[8,9],[2,3]]
# X_test : [[4,5]]

# 데이터(x)와 레이블(y)을 넣었을 경우
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33,
random_state=321)
# X_train : [[4,5],[0,1],[6,7]]
# Y_train : [2,0,3]
# X_test : [[2,3],[8,9]]
# Y_test : [1,4]
```