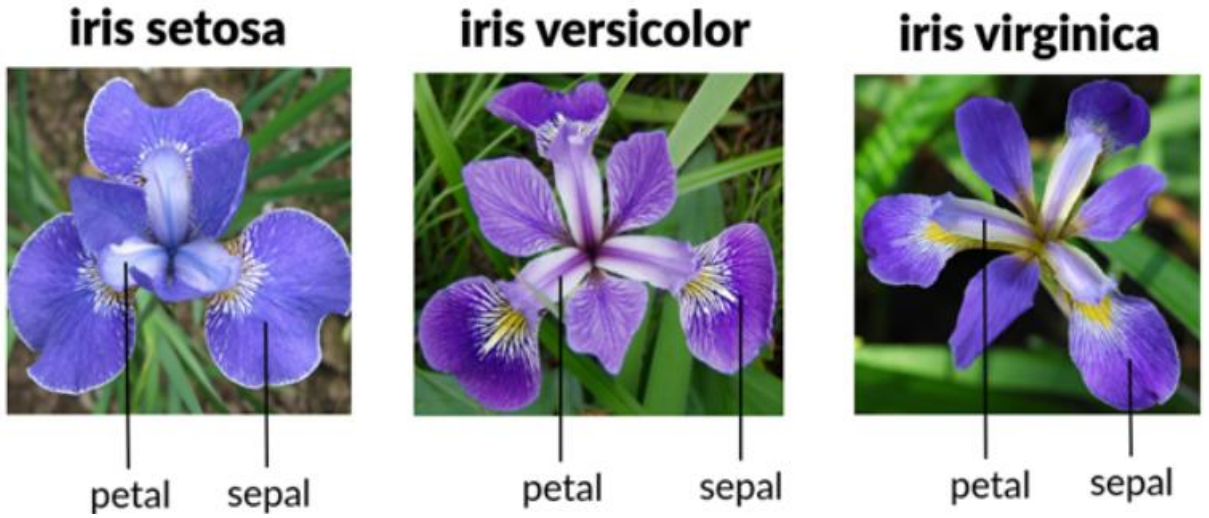
딥러닝 분류모델

최석재 *lingua@naver.com*

딥러닝 다중분류

딥러닝 다중분류

- iris 품종 구분하기
  - 아이리스 데이터를 이용하여 세부 품종을 구분한다
  - setosa, versicolor, virginica 의 3종이 있다



딥러닝 다중분류

- 데이터 불러오기

```
from google.colab import drive
drive.mount('/content/gdrive')

# 경로 변경
%cd /content/gdrive/MyDrive/pytest_img/

import pandas as pd

df = pd.read_csv("iris.csv")
df.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

딥러닝 다중분류

- 데이터 뒤섞기

```
df = df.sample(frac=1)    # 원본 데이터의 100%를 랜덤하게 불러온다
df.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
42	4.4	3.2	1.3	0.2	setosa
96	5.7	2.9	4.2	1.3	versicolor
53	5.5	2.3	4.0	1.3	versicolor
130	7.4	2.8	6.1	1.9	virginica
103	6.3	2.9	5.6	1.8	virginica

딥러닝 다중분류

- 데이터 확인

```
print(df.info())
```

```
print(df.describe())
```

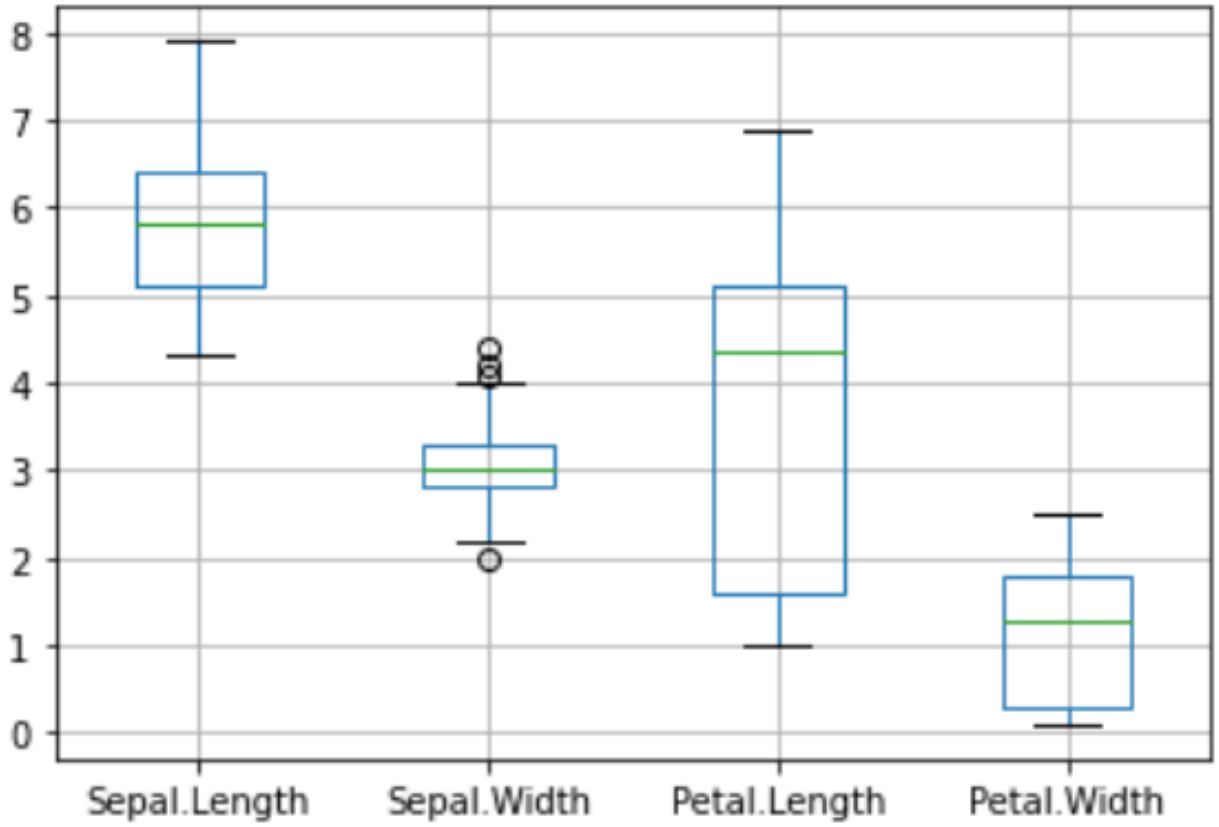
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 42 to 132
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Sepal.Length  150 non-null    float64
 1   Sepal.Width   150 non-null    float64
 2   Petal.Length  150 non-null    float64
 3   Petal.Width   150 non-null    float64
 4   Species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
None
```

◀ 종속변수가 문자열이다

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

딥러닝 다중분류

- 데이터 확인



이상치는 거의 없다

딥러닝 다중분류

```
# 독립변수와 종속변수 분리
X = df.iloc[:, 0:4]
y = df.iloc[:, 4]


# 훈련 데이터와 테스트 데이터 분리
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)

print(X_train.head())
print("**********")
print(X_test.head())
```

```
     Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
118           7.7          2.6           6.9          2.3
15            5.7          4.4           1.5          0.4
92            5.8          2.6           4.0          1.2
98            5.1          2.5           3.0          1.1
76            6.8          2.8           4.8          1.4
**********
     Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
123           6.3          2.7           4.9          1.8
22            4.6          3.6           1.0          0.2
36            5.5          3.5           1.3          0.2
47            4.6          3.2           1.4          0.2
67            5.8          2.7           4.1          1.0
```

딥러닝 다중분류

- 종속변수 확인
  - 문자열로 되어 있음을 확인한다
  - 이것을 숫자형으로 변환한 뒤, 원-핫 인코딩을 한다

```
print(y_train)
```

```
118     virginica
15         setosa
92     versicolor
98     versicolor
76     versicolor
          ...
149     virginica
148     virginica
140     virginica
72     versicolor
88     versicolor
Name: Species, Length: 112, dtype: object
```

딥러닝 다중분류

- 문자열을 숫자형으로 변환
  - to_categorical 함수로 원-핫 인코딩을 하기 위하여
  - 먼저 LabelEncoder로 숫자형으로 변환
  - 결과는 0부터 시작하여 분류 개수가 된다

```
from sklearn.preprocessing import LabelEncoder
e = LabelEncoder()
e.fit(y_train)                    # 훈련 데이터가 아닌 전체 데이터셋을 넣어도 된다
y_train = e.transform(y_train)

print("y_train: ", y_train)
```

```
y_train:  [2 0 1 1 1 1 0 0 2 1 0 0 1 1 1 0 0 0 1 0 1 1 0 2 2 2 1 1 2 1 2 0 0 2 1 0 2
 2 2 2 2 0 2 0 1 2 2 2 0 1 0 0 0 1 1 1 2 2 1 2 1 1 0 2 1 0 1 2 2 1 2 1 2 1
 1 2 1 0 1 1 0 0 0 2 2 0 2 0 0 0 2 0 1 1 1 2 1 0 2 0 0 2 1 2 0 2 1 2 2 2 1
 1]
```

pd.get_dummies()는 숫자와 문자를 모두 받을 수 있지만,
여기에서 사용할 to_categorical()은 숫자형만 받는다

딥러닝 다중분류

- 변환 결과 확인
  - 어떤 문자열이 어떤 숫자로 변환되었는지를 확인한다
  - 분류 종류로 'setosa, versicolor, virginica'가 있었으므로 이를 변환해본다

```
# 변환 결과를 확인한다
y_train_transformed = e.transform(["setosa", "versicolor", "virginica"])
print(e.classes_[0], y_train_transformed[0], ",", e.classes_[1], y_train_transformed[1],
",", e.classes_[2], y_train_transformed[2])
```

```
setosa 0 , versicolor 1 , virginica 2
```

정렬 순이므로 순서가 바뀌지는 않는다

e.classes_ 를 출력하여도 된다

```
e.classes_
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

딥러닝 다중분류

- 원-핫 인코딩

```
# 0, 1, 2 중 하나를 예측하는 다중분류 상황이므로 원-핫 인코딩
import tensorflow as tf
y_train = tf.keras.utils.to_categorical(y_train)
print(y_train)
```

```
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
```

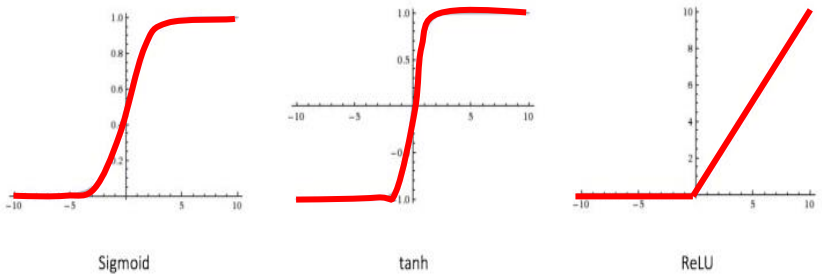
- 출력층이 0~2번 자리 중 하나를 차지하게 만들었다
- 다중분류는 각 분류의 확률을 계산해야 하므로 원-핫 인코딩이 필요하다

딥러닝 다중분류

- 모델 설정

```
# 해당 클래스를 직접 사용할 수 있도록 import 한다
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(16, input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))
```



conceptually,

sigmoid: $P(class1)$

softmax: $\frac{P(classN)}{P(class1)+P(class2)+P(class3)+\cdots}$

딥러닝 다중분류

- 모델 컴파일 및 훈련

데이터가 얼마되지 않아 배치 사이즈를 1로 하였다
배치 사이즈는 일반적으로 8~512를 사용한다. 기본값은 None으로 되어 있으나,
이 값을 넣지 않으면 기본값 32가 들어간다

```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc'])
model.fit(X_train, y_train, epochs=50, batch_size=1)
```

```
Epoch 1/50
112/112 [==============================] - 1s 3ms/step - loss: 1.0342 - acc: 0.4732
Epoch 2/50
112/112 [==============================] - 0s 3ms/step - loss: 0.7495 - acc: 0.6875
Epoch 3/50
112/112 [==============================] - 0s 3ms/step - loss: 0.6483 - acc: 0.6964
Epoch 4/50
112/112 [==============================] - 0s 3ms/step - loss: 0.5872 - acc: 0.7500
Epoch 5/50
112/112 [==============================] - 0s 4ms/step - loss: 0.5397 - acc: 0.7321
Epoch 6/50
112/112 [==============================] - 0s 3ms/step - loss: 0.5129 - acc: 0.7500
Epoch 7/50
112/112 [==============================] - 0s 4ms/step - loss: 0.4880 - acc: 0.7321
```

딥러닝 다중분류

테스트 데이터에도 똑같은 종류가 들어가 있다면
e = LabelEncoder() 및 e.fit()을 다시 해도 0, 1, 2의 순서가 바뀌지는 않는다
그러나 같은 인코더로 재활용하는 것이 좋다

저장과 불러오기는 다음의 방법으로 한다
Train>
encoder = LabelEncoder()
encoder.fit(X)
numpy.save('classes.npy', encoder.classes_)

Test>
encoder = LabelEncoder()
encoder.classes_ = numpy.load('classes.npy')

- 테스트 데이터 원-핫 인코딩
  - 테스트 데이터의 종속변수도 원-핫 인코딩을 해야 한다

```
# e.fit(y_test)  하면 안됨!
y_test = e.transform(y_test)
print(y_test)
```

```
[2 0 0 0 1 2 2 0 1 0 2 2 2 1 0 1 2 2 2 0 2 1 0 2 0 1 1 0 0 0 1 0 0 2 0 1 1
 0]
```

```
# 0, 1, 2 중 하나를 예측하는 다중분류 상황이므로 원-핫 인코딩
import tensorflow as tf
y_test = tf.keras.utils.to_categorical(y_test)
print(y_test)
```

```
[[0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
```

딥러닝 다중분류

- 성능 평가

```
print("Test Data Accuracy: ", model.evaluate(X_test, y_test)[1])
```

```
2/2 [==============================] - 0s 10ms/step - loss: 0.1099 - acc: 1.0000
Test Data Accuracy:  1.0
```

딥러닝 다중분류

- 모델 저장 및 재사용

```
# 경로 변경
import os
os.chdir("/content/gdrive/MyDrive/pytest_img/models")

# 모델 저장
model.save("iris.h5")

# 모델 불러오기
from keras.models import load_model
loaded_model = load_model("iris.h5")

# 불러온 모델로 전체 테스트 데이터 예측
predictions = loaded_model.predict(X_test)
```

딥러닝 다중분류

- 예측 결과 확인

- print(predictions)

```
[[9.99821246e-01 1.78828035e-04 7.25226823e-11]
 [9.99945164e-01 5.48509197e-05 1.16909919e-11]
 [3.29550893e-08 2.23162044e-02 9.77683783e-01]
 [2.98653194e-05 7.87513137e-01 2.12456986e-01]
 [6.18639821e-03 9.89872336e-01 3.94123048e-03]
 [1.61957578e-04 9.36646223e-01 6.31919280e-02]
 [5.64381371e-05 8.86721313e-01 1.13222212e-01]
 [9.98869121e-01 1.13081443e-03 3.20351989e-09]
 [8.11888549e-06 3.79088104e-01 6.20903790e-01]
 [9.98180389e-01 1.81967032e-03 3.25285332e-09]
 [8.05882564e-07 1.07132569e-01 8.92866671e-01]
 [9.99994278e-01 5.76642378e-06 1.19786532e-13]
 [1.94435968e-04 9.62077320e-01 3.77282426e-02]
 [4.61347872e-06 4.33459103e-01 5.66536248e-01]
 [9.99824703e-01 1.75291367e-04 5.85798354e-11]
 [2.97597808e-06 2.78409839e-01 7.21587241e-01]
 [1.56627891e-07 1.79043505e-02 9.82095480e-01]
 [9.99716818e-01 2.83200003e-04 3.06374537e-10]
 [9.00281043e-01 7.19034571e-04 1.08310600e-00]
```

딥러닝 다중분류

- 카테고리 분류

- import numpy as np
- predicted_classes = np.argmax(predictions[:10], axis=1)
- print("예측 결과:", predicted_classes)

```
예측 결과: [0 0 2 1 1 1 1 0 2 0]
```

딥러닝 다중분류

- 1개 데이터 예측

```
new_data = {"Sepal.Length":[5.3], "Sepal.Width":[3.4], "Petal.Length":[1.4], "Petal.Width":[0.2]}
new_data = pd.DataFrame(new_data)
print(new_data)

result = loaded_model.predict(new_data)
print("result: ", result)
```

```
   Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
0           5.3          3.4           1.4          0.2
1/1 [==============================] - 0s 24ms/step
result:  [[9.9936825e-01 6.3171046e-04 4.2932875e-09]]
```

먼저 딕셔너리로 1개 데이터를 만들고, 이를 데이터프레임으로 변환

딥러닝 다중분류

- 1개 데이터 예측결과 변환
  - 결과가 숫자형으로 되어 있으므로 이를 다시 문자형으로 변환한다
  - e.transform() 또는 e.classes_ 의 결과를 참고하여 코딩한다

```
import numpy as np
result_idx = np.argmax(result)

if result_idx == 0:
    print("setosa")
elif result_idx == 1:
    print("versicolor")
elif result_idx == 2:
    print("virginica")
```

또는 e.inverse_transform([result_idx])[0]

```
setosa
```