● 폐암 수술 환자의 생존율 예측 딥러닝

ThoracicSurgery.csv
폴란드의 브로츠와프 의과대학에서 2013년 공개한 폐암 수술 환자의
수술 전 진단 데이터 x 와 수술 후 생존 결과 y 를 기록한 실제 의료 기록 데이터

1. 딥러닝에 필요한 라이브러리 임포트하기

In [1]:

```
# 딥러닝을 구동하는 데 필요한 케라스 함수를 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

# 필요한 라이브러리를 불러옵니다.
import numpy as np
import tensorflow as tf
```

2. 준비된 데이터셋(빅데이터)을 numpy 기반으로 읽어오기

In [2]:

```
# 준비된 수술 환자 데이터를 불러들입니다.
Data_set = np.loadtxt("./dataset/ThoracicSurgery.csv", delimiter=",")
```

In [3]:

```
Data_set
```

Out[3]:

```
array([[293.  ,   1.  ,   3.8 , ...,   0.  ,  62.  ,   0.  ],
       [  1.  ,   2.  ,   2.88, ...,   0.  ,  60.  ,   0.  ],
       [  8.  ,   2.  ,   3.19, ...,   0.  ,  66.  ,   1.  ],
       ...,
       [406.  ,   6.  ,   5.36, ...,   0.  ,  62.  ,   0.  ],
       [ 25.  ,   8.  ,   4.32, ...,   0.  ,  58.  ,   1.  ],
       [447.  ,   8.  ,   5.2 , ...,   0.  ,  49.  ,   0.  ]])
```

In [4]:

```
type(Data_set)
```

Out[4]:

```
numpy.ndarray
```

In [5]:

```
np.shape(Data_set)
```

Out[5]:

(470, 18)

In [6]:

```
# 환자의 기록과 수술 결과를 X와 Y로 구분하여 저장합니다.
X = Data_set[:,0:17]
Y = Data_set[:,17]
```

3. 읽어온 데이터셋을 학습용과 테스트용으로 분리하기

from sklearn.model_selection import train_test_split 제공되는 train_test_split( ) 기능 활용

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

In [7]:

```
# 학습 데이터 70 %, 테스트 데이터셋 30% 로 설정하기
seed = 0
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=seed)
```

In [8]:

```
print('● type(X_train) : ', type(X_train)) # train_test_split( ) 함수의 리턴 값은 numpy.ndarra
print('● np.shape(X_train) : ',  np.shape(X_train)) # 학습 데이터 모양
print('● len(X_train) : ', len(X_train)) # 학습 데이터 개수
print(X_train)  #학습 데이터 속성 확인
```

```
● type(X_train) :  <class 'numpy.ndarray'>
● np.shape(X_train) :  (329, 17)
● len(X_train) :  329
[[434.    3.    4.28 ...   1.    0.   66.  ]
 [396.    4.    3.04 ...   1.    0.   64.  ]
 [ 26.    5.    4.56 ...   1.    0.   57.  ]
 ...
 [ 85.    3.    4.28 ...   1.    0.   51.  ]
 [419.    2.    2.6  ...   0.    0.   70.  ]
 [161.    3.    2.92 ...   0.    0.   76.  ]]
```

In [9]:

```
print('● type(X_test) : ', type(X_test)) # train_test_split( ) 함수의 리턴 값은 numpy.ndarray
print('● np.shape(X_test) : ',  np.shape(X_test)) # 테스트 데이터 모양
print('● len(X_test) : ', len(X_test)) # 테스트 데이터 개수
print(X_test)  #테스트 데이터 속성 확인
```

```
● type(X_test) :  <class 'numpy.ndarray'>
● np.shape(X_test) :  (141, 17)
● len(X_test) :  141
[[344.     3.     2.96 ...   1.     0.    60.  ]
 [238.     3.     3.24 ...   1.     0.    69.  ]
 [172.     2.     2.88 ...   1.     0.    62.  ]
 ...
 [ 40.     3.     4.6  ...   1.     0.    52.  ]
 [456.     4.     2.92 ...   1.     0.    70.  ]
 [355.     3.     4.28 ...   1.     0.    60.  ]]
```

In [10]:

```
print('● type(Y_train) : ', type(Y_train)) # train_test_split( ) 함수의 리턴 값은 numpy.ndarra
print('● np.shape(Y_train) : ',  np.shape(Y_train)) # 학습 데이터 클래스 모양
print('● len(Y_train) : ', len(Y_train)) # 학습 데이터 클래스 개수
print(Y_train)  #학습 데이터 클래스 속성 확인
```

```
● type(Y_train) :  <class 'numpy.ndarray'>
● np.shape(Y_train) :  (329,)
● len(Y_train) :  329
[0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0.
 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0.
 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1.
 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
```

In [11]:

```
print('● type(Y_test) : ', type(Y_test)) # train_test_split( ) 함수의 리턴 값은 numpy.ndarray
print('● np.shape(Y_test) : ',  np.shape(Y_test)) # 테스트 데이터 클래스 모양
print('● len(Y_test) : ', len(Y_test)) # 테스트 데이터 클래스 개수
print(Y_test)  #테스트 데이터 클래스 속성 확인
```

```
● type(Y_test) :  <class 'numpy.ndarray'>
● np.shape(Y_test) :  (141,)
● len(Y_test) :  141
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0.
 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0.]
```

4. 딥러닝 모델 설계하기

keras에서 제공되는 Sequential()의 add()활용하여 모델 설계

In [12]:

```
# 딥러닝 구조를 결정합니다(모델을 설정하는 부분)

model = Sequential() #keras에서 제공되는 기능

#1번째 층 : 입력 x는 17개, 출력은 30개, 활성화함수 relu
model.add(Dense(30, input_dim=17, activation='relu'))

#2번째 층 : 입력 x는 30개, 출력은 1개, 활성화함수 sigmoid
model.add(Dense(1, activation='sigmoid'))
```

In [13]:

```
model
```

Out[13]:

```
<keras.engine.sequential.Sequential at 0x27d710fbb20>
```

5. 딥러닝 모델 컴파일 및 실행하기

keras의 Sequential()에서 제공하는 compile( ) 기능으로 loss, optimizer, metrics 설정

In [14]:

```
#옵션 중에서 loss='mean_squared_error'는 선형회귀에서 사용
#model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])

#옵션 중에서 loss='binary_crossentropy'는 시그모이드를 이용한 2진 분류에서 사용
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

keras의 Sequential()에서 제공하는 fit( ) 기능으로 x, y, 학습 반복횟수, 배치 사이즈 설정

In [15]:

```
#batch_size=1로 하면, 329개의 데이터를 1개씩 처리하므로 실행 결과에서 329/329 출력
#model.fit(X_train, Y_train, epochs=500, batch_size=1) # 학습 데이터셋 329 개로 학습

#batch_size=10 으로 하면, 329개의 데이터를 10개씩 모아서 처리하므로 실행 결과에서 33/33으로 출력
model.fit(X_train, Y_train, epochs=500, batch_size=10) # 학습 데이터셋 329 개로 학습
```

```
Epoch 44/500
33/33 [==============================] - 0s 1ms/step - loss: 0.4333 - accuracy:
0.8298
Epoch 45/500
33/33 [==============================] - 0s 1ms/step - loss: 0.4418 - accuracy:
0.8359
Epoch 46/500
33/33 [==============================] - 0s 2ms/step - loss: 0.4235 - accuracy:
0.8359
Epoch 47/500
33/33 [==============================] - 0s 2ms/step - loss: 0.4978 - accuracy:
0.8328
Epoch 48/500
33/33 [==============================] - 0s 2ms/step - loss: 0.4907 - accuracy:
0.8024
Epoch 49/500
33/33 [==============================] - 0s 1ms/step - loss: 0.4416 - accuracy:
0.8389
Epoch 50/500
33/33 [==============================] - 0s 1ms/step - loss: 0.4681 - accuracy:
```

6. 테스트 데이터를 기반으로 딥러닝 평가하기

keras의 Sequential()에서 제공하는 model.evaluate( ) 기능 사용
리턴 값 : [loss 오차 , acc 정확도]

In [16]:

```
print('● 테스트 데이터 개수 : ', len(X_test), '개')
ev = model.evaluate(X_test, Y_test, batch_size=1) #디폴트 출력 내용 loss, accuracy 있음
print('● [ loss 오차 , accuracy 정확도] =' , ev)
```

```
● 테스트 데이터 개수 :  141 개
141/141 [==============================] - 0s 1ms/step - loss: 0.6554 - accuracy: 0.
8156
● [ loss 오차 , accuracy 정확도] = [0.6554419994354248, 0.8156028389930725]
```

In [17]:

```
# 30%에 해당하는 141개의 테스트 데이터셋으로 정확도 계산 결과를 출력
# 테스트 데이터셋 141 개로 정확도 계산
# 리턴 값 : [ loss 오차 , acc 정확도] 중에서 Loss 정확도 부분만 출력

print("₩n Test Loss : %.4f %%" % (ev[0]))
```

```
Test Loss : 0.6554 %
```

In [18]:

```
# 30%에 해당하는 141개의 테스트 데이터셋으로 정확도 계산 결과를 출력
# 테스트 데이터셋 141 개로 정확도 계산
# 리턴 값 : [ loss 오차 , acc 정확도] 중에서 Accuracy 정확도 부분만 출력

print("₩n Test Accuracy : %.4f %%" % (ev[1]))
```

```
 Test Accuracy : 0.8156 %
```

<참고>
밀리 초, millisecond (ms, msec)는 10-3(1000분의 1)초에 해당하는 시간의 단위
마이크로 초, microsecond (us)는 10-6(100만분의 1)초에 해당하는 시간의 단위
피코 초, picosecond는 10-12(1조분의 1)초에 해당하는 시간의 단위
펨토 초, femtosecond는 10-15(1000조분의 1)초에 해당하는 시간의 단위
아토 초, attosecond는 10-18초에 해당하는 시간의 단위

7. 새로운 데이터를 대상으로 딥러닝 모델을 사용하여 예측하기

keras의 Sequential()에서 제공하는 predict() 기능 사용
predict()은 학습 및 테스트에 사용하지 않은 데이터를 사용하는 것이 바람직..

In [19]:

```
#kim 입력 데이터

kim = np.array([[293,1,3.8,2.8,0,0,0,0,0,0,12,0,0,0,1,0,62]])
test_kim = model.predict(kim)

print("kim 생존율 예측 결과 : ", test_kim)
test_kim  = test_kim*100
print("kim 생존율 예측 결과(백분율) : ", test_kim, '%')
print("kim 폐암 수술 후 생존율 예측 : %.2f" %test_kim, "%")
```

```
1/1 [==============================] - 0s 113ms/step
kim 생존율 예측 결과 :  [[0.06094453]]
kim 생존율 예측 결과(백분율) :  [[6.094453]] %
kim 폐암 수술 후 생존율 예측 : 6.09 %
```

In [20]:

```
#park 입력 데이터

park = np.array([[363,5,2.38,1.72,1,0,1,0,1,0,12,1,0,1,1,0,87]])
test_park = model.predict(park)

print("park 생존율 예측 결과 : ", test_park)
park_kim  = test_kim*100
print("park 생존율 예측 결과(백분율) : ", test_park, '%')
print("park 폐암 수술 후 생존율 예측 : %.2f" %test_park, "%")
```

```
1/1 [==============================] - 0s 28ms/step
park 생존율 예측 결과 :  [[0.77531904]]
park 생존율 예측 결과(백분율) :  [[0.77531904]] %
park 폐암 수술 후 생존율 예측 : 0.78 %
```

In [21]:

```
#lee 입력 데이터

lee = np.array([[25,8,4.32,3.2,0,0,0,0,0,0,11,0,0,0,0,0,58]])
test_lee = model.predict(lee)

print("lee 생존율 예측 결과 : ", test_lee)
park_lee  = test_lee*100
print("lee 생존율 예측 결과(백분율) : ", test_lee, '%')
print("lee 폐암 수술 후 생존율 예측 : %.2f" %test_lee, "%")
```

```
1/1 [==============================] - 0s 28ms/step
lee 생존율 예측 결과 :  [[0.81676227]]
lee 생존율 예측 결과(백분율) :  [[0.81676227]] %
lee 폐암 수술 후 생존율 예측 : 0.82 %
```

8. 딥러닝 모델 저장하기

keras의 Sequential()에서 제공하는 save() 기능 사용
h5 형식으로 학습 결과 저장

In [22]:

```
model.save('DL_RESULT/ThoracicSurgery.h5')
```

numpy 참고 자료

In [23]:

```
# 랜덤값 출력 (참고)
np.random.seed(0) #seed값을 설정하면 실행 시, 매번 동일한 랜덤값을 출력
print(np.random.random(3))

np.random.seed(2) #seed값을 설정하면 실행 시, 매번 동일한 랜덤값을 출력
print(np.random.random(3))

np.random.seed(0) #seed값을 설정하면 실행 시, 매번 동일한 랜덤값을 출력
print(np.random.random(3))
```

```
[0.5488135  0.71518937 0.60276338]
[0.4359949  0.02592623 0.54966248]
[0.5488135  0.71518937 0.60276338]
```

In [24]:

```
print(kim)
print(np.shape(kim))
print('='*50)

print(park)
print(np.shape(park))
print('='*50)

print(lee)
print(np.shape(lee))
print('='*50)
```

```
[[293.    1.    3.8   2.8   0.    0.    0.    0.    0.    0.   12.    0.
    0.    0.    1.    0.   62. ]]
(1, 17)
==================================================
[[363.    5.    2.38  1.72  1.    0.    1.    0.    1.    0.
   12.    1.    0.    1.    1.    0.   87.  ]]
(1, 17)
==================================================
[[25.    8.    4.32  3.2   0.    0.    0.    0.    0.    0.   11.    0.
   0.    0.    0.    0.   58.  ]]
(1, 17)
==================================================
```

h5py 참고 자료

In [25]:

```
import h5py
f1 = h5py.File("DL_RESULT/ThoracicSurgery.h5", 'r')

print(f1)
print(list(f1.keys()))
print(list(f1.values()))
print(list(f1.attrs.keys()))
print(list(f1.attrs.values()))
```

```
<HDF5 file "ThoracicSurgery.h5" (mode r)>
['model_weights', 'optimizer_weights']
[<HDF5 group "/model_weights" (3 members)>, <HDF5 group "/optimizer_weights" (1 memb
ers)>]
['backend', 'keras_version', 'model_config', 'training_config']
['tensorflow', '2.9.0', '{"class_name": "Sequential", "config": {"name": "sequentia
l", "layers": [{"class_name": "InputLayer", "config": {"batch_input_shape": [null, 1
7], "dtype": "float32", "sparse": false, "ragged": false, "name": "dense_input"}},
{"class_name": "Dense", "config": {"name": "dense", "trainable": true, "batch_input_
shape": [null, 17], "dtype": "float32", "units": 30, "activation": "relu", "use_bia
s": true, "kernel_initializer": {"class_name": "GlorotUniform", "config": {"seed": n
ull}}, "bias_initializer": {"class_name": "Zeros", "config": {}}, "kernel_regularize
r": null, "bias_regularizer": null, "activity_regularizer": null, "kernel_constrain
t": null, "bias_constraint": null}}, {"class_name": "Dense", "config": {"name": "den
se_1", "trainable": true, "dtype": "float32", "units": 1, "activation": "sigmoid",
"use_bias": true, "kernel_initializer": {"class_name": "GlorotUniform", "config":
{"seed": null}}, "bias_initializer": {"class_name": "Zeros", "config": {}}, "kernel_
regularizer": null, "bias_regularizer": null, "activity_regularizer": null, "kernel_
constraint": null, "bias_constraint": null}}]}}', '{"loss": "binary_crossentropy",
"metrics": [[{"class_name": "MeanMetricWrapper", "config": {"name": "accuracy", "dty
pe": "float32", "fn": "binary_accuracy"}}]], "weighted_metrics": null, "loss_weight
s": null, "optimizer_config": {"class_name": "Adam", "config": {"name": "Adam", "lea
rning_rate": 0.0010000000474974513, "decay": 0.0, "beta_1": 0.8999999761581421, "bet
a_2": 0.9990000128746033, "epsilon": 1e-07, "amsgrad": false}}}']
```

In [ ]:

```
```