사례2: 당뇨병 발병 예측

sydong@sookmyung.ac.kr

당뇨병 발병 예측하기

- 목표
  - ✓ 딥러닝과 그리드 탐색 방법을 사용해 당뇨병 발병을 예측하는 사례를 소개한다
  - ✓ 또한 딥러닝의 개념과 최적화를 위한 파라미터를 소개하고, 적절한 파라미터를 선택하는 방법도 다룬다
- 순서
  - ✓ 준비하기
  - ✓ 케라스 모델 설계
  - ✓ 그리드 탐색 실행하기
  - ✓ 드롭아웃 규제로 과적합 줄이기
  - ✓ 최적의 초매개변수 찾기
  - ✓ 최적 초매개변수를 사용해 예측하기

준비하기

- 필요한 패키지들을 로드하고 데이터셋을 불러온다
  - ✓ 요약 통계량 확인
    - 데이터셋 문제점?

```
7. For Each Attribute: (all numeric-valued)
   1. Number of times pregnant
   2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
   3. Diastolic blood pressure (mm Hg)
   4. Triceps skin fold thickness (mm)
   5. 2-Hour serum insulin (mu U/ml)
   6. Body mass index (weight in kg/(height in m)^2)
   7. Diabetes pedigree function
   8. Age (years)
   9. Class variable (0 or 1)
```

*각 열의 의미

```
# import libraries
import pandas as pd
import numpy as np
import sklearn
from tensorflow import keras
```

```
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['n_pregnant','glucose_concentration','blood_pressure (mm Hg)', 'skin_thickness (mm)',
         'serum_insulin (mu U/ml)','BMI','pedigree_function','age','class']
df = pd.read_csv(url, names=names)
```

```
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
n_pregnant	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
glucose_concentration	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
blood_pressure (mm Hg)	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
skin_thickness (mm)	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
serum_insulin (mu U/ml)	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
pedigree_function	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
class	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

데이터 전처리

- 결측값 처리
  - ✓ 0을 NaN으로 바꾸기
- 결측값있는 행 제거
- 데이터셋을 NumPy 배열로 변환
  - ✓ df.values

- 최종 데이터셋 크기는?

```
cols = ['glucose_concentration','blood_pressure (mm Hg)','skin_thickness (mm)','serum_insulin (mu U/ml)', 'BMI']
for col in cols:
  df[col].replace(0, np.nan, inplace=True)
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
n_pregnant	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
glucose_concentration	763.0	121.686763	30.535641	44.000	99.00000	117.0000	141.00000	199.00
blood_pressure (mm Hg)	733.0	72.405184	12.382158	24.000	64.00000	72.0000	80.00000	122.00
skin_thickness (mm)	541.0	29.153420	10.476982	7.000	22.00000	29.0000	36.00000	99.00
serum_insulin (mu U/ml)	394.0	155.548223	118.775855	14.000	76.25000	125.0000	190.00000	846.00
BMI	757.0	32.457464	6.924988	18.200	27.50000	32.3000	36.60000	67.10
pedigree_function	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
class	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

```
df.dropna(inplace=True)
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
n_pregnant	392.0	3.301020	3.211424	0.000	1.00000	2.0000	5.000	17.00
glucose_concentration	392.0	122.627551	30.860781	56.000	99.00000	119.0000	143.000	198.00
blood_pressure (mm Hg)	392.0	70.663265	12.496092	24.000	62.00000	70.0000	78.000	110.00
skin_thickness (mm)	392.0	29.145408	10.516424	7.000	21.00000	29.0000	37.000	63.00
serum_insulin (mu U/ml)	392.0	156.056122	118.841690	14.000	76.75000	125.5000	190.000	846.00
BMI	392.0	33.086224	7.027659	18.200	28.40000	33.2000	37.100	67.10
pedigree_function	392.0	0.523046	0.345488	0.085	0.26975	0.4495	0.687	2.42
age	392.0	30.864796	10.200777	21.000	23.00000	27.0000	36.000	81.00
class	392.0	0.331633	0.471401	0.000	0.00000	0.0000	1.000	1.00

```
dataset = df.values
print(dataset.shape)
```

```
(392, 9)
```

데이터 전처리

- 데이터셋 분리
  - ✓ 입력 데이터셋 X와 타깃 값 Y로 나누기
  - ✓ 값을 정수로 바꾸기
    - .astype(int)
  - ✓ 데이터셋 형태 확인
    - 처음 다섯 케이스 출력
- 데이터 정규화
  - ✓ sklearn에서 제공하는 StandardScaler
    - 평균 0, 표준편차 1을 가짐
    - 머신러닝 알고리즘들이 모든 열에 대해 특별히 의존하거나 가중치를 부여하는 일 없도록 균등하게 처리할 수 있게 하는 것

```
X = dataset[:, 0:8]
Y = dataset[:, 8].astype(int)
```

```
print(X.shape)
print(Y.shape)
print(X[:5])
print(Y[:5])
```

```
(392, 8)
(392,)
[[1.000e+00 8.900e+01 6.600e+01 2.300e+01 9.400e+01 2.810e+01 1.670e-01
  2.100e+01]
 [0.000e+00 1.370e+02 4.000e+01 3.500e+01 1.680e+02 4.310e+01 2.288e+00
  3.300e+01]
 [3.000e+00 7.800e+01 5.000e+01 3.200e+01 8.800e+01 3.100e+01 2.480e-01
  2.600e+01]
 [2.000e+00 1.970e+02 7.000e+01 4.500e+01 5.430e+02 3.050e+01 1.580e-01
  5.300e+01]
 [1.000e+00 1.890e+02 6.000e+01 2.300e+01 8.460e+02 3.010e+01 3.980e-01
  5.900e+01]]
[0 1 1 1 1]
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X)
```

```
X_standardized = scaler.transform(X)
data = pd.DataFrame(X_standardized)
data.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
0	392.0	-9.063045e-18	1.001278	-1.029213	-0.717427	-0.405640	0.529718	4.271153
1	392.0	1.132881e-17	1.001278	-2.161731	-0.766596	-0.117696	0.660984	2.445459
2	392.0	-4.531523e-16	1.001278	-3.739001	-0.694164	-0.053146	0.587873	3.151946
3	392.0	1.087565e-16	1.001278	-2.108484	-0.775531	-0.013844	0.747843	3.223325
4	392.0	1.064908e-16	1.001278	-1.196867	-0.668179	-0.257445	0.285988	5.812990
5	392.0	1.631348e-16	1.001278	-2.120941	-0.667678	0.016210	0.571870	4.846172
6	392.0	1.812609e-17	1.001278	-1.269525	-0.734091	-0.213147	0.475164	5.497667
7	392.0	1.110223e-16	1.001278	-0.968299	-0.771985	-0.379357	0.504056	4.921123

케라스 모델 정의

- 모델 정의 과정을 create_model()이라는 사용자 정의 함수로 만듦
  - ✓ 매개변수를 바꿔가면서 모델을 여러 번 초기화하여 재사용할 것이기 때문에 함수화 하는 것이 유리함
- model.summary()로 모델의 특징을 확인할 수 있다
  - ✓ 이 신경망의 모든 매개변수의 개수는 113개

```
import tensorflow as tf
from tensorflow import keras

from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV, KFold
```

```
seed = 6
np.random.seed(seed)

def create_model():
  model = keras.models.Sequential()
  model.add(keras.layers.Dense(8, input_dim = 8, kernel_initializer='normal', activation='relu'))
  model.add(keras.layers.Dense(4, kernel_initializer='normal', activation='relu'))
  model.add(keras.layers.Dense(1, activation='sigmoid'))
  adam = keras.optimizers.Adam(learning_rate=0.01)

  model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])
  return model
```

```
model = create_model()
print(model.summary())
```

```
Model: "sequential_71"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_213 (Dense)           (None, 8)                 72

 dense_214 (Dense)           (None, 4)                 36

 dense_215 (Dense)           (None, 1)                 5

=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
None
```

그리드 탐색 실행하기

- Keras 모델은 SciKeras 모듈의 KerasClassifier 클래스로 래핑(wrapping)하여 사이킷런에서 사용가능
  - ✓ 매개변수 model은 모델을 정의하는 데 사용되는 함수를 지정
- 그리드 탐색 매개변수 설정
  - ✓ 배치 크기와 에포크
  - ✓ 그리드 검색 알고리즘에 넣기 위해 dict()로 파이썬 딕셔너리로 바꿔줌

```
model = KerasClassifier(model=create_model)
```

```
batch_size = [10, 20, 40]
epochs = [10, 50, 100]

param_grid = dict(batch_size = batch_size, epochs=epochs)
```

```
grid = GridSearchCV(estimator=model, param_grid=param_grid,
                    cv=KFold(random_state=seed, shuffle=True), verbose=0)
grid_results=grid.fit(X_standardized, Y)
```

```
Epoch 1/10
32/32 [==============================] - 1s 3ms/step - loss: 0.6542 - accuracy: 0.6805
Epoch 2/10
32/32 [==============================] - 0s 3ms/step - loss: 0.5079 - accuracy: 0.7636
Epoch 3/10
32/32 [==============================] - 0s 4ms/step - loss: 0.4658 - accuracy: 0.7764
Epoch 4/10
32/32 [==============================] - 0s 6ms/step - loss: 0.4552 - accuracy: 0.7827
Epoch 5/10
32/32 [==============================] - 0s 3ms/step - loss: 0.4456 - accuracy: 0.7764
Epoch 6/10
32/32 [==============================] - 0s 2ms/step - loss: 0.4358 - accuracy: 0.7827
Epoch 7/10
32/32 [==============================] - 0s 3ms/step - loss: 0.4329 - accuracy: 0.7891
```

그리드 탐색 실행하기

- 결과를 보고하는 코드
  - ✓ 테스트 데이터에 대해 78.59%의 정확도를 보임
    - 훈련 성능에 비해 떨어지는 것은 새로운 데이터에 대한 일반화가 제대로 안되고 있음을 나타냄
    - 훈련데이터에 대한 과적합이 발생한 것

```
print("Best: {0}, **using** {1}".format(grid_results.best_score_, grid_results.best_params_))
means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
Best: 0.7859136643946771, **using** {'batch_size': 40, 'epochs': 50}
0.7601752677702045 (0.0261691550267947 6) with: {'batch_size': 10, 'epochs': 10}
0.7680298604349236 (0.0352772742292394 5) with: {'batch_size': 10, 'epochs': 50}
0.7093476144109057 (0.0619703393929477 3) with: {'batch_size': 10, 'epochs': 100}
0.772963323596235 (0.034341279422125594) with: {'batch_size': 20, 'epochs': 10}
0.7679324894514767 (0.036700379636644326) with: {'batch_size': 20, 'epochs': 50}
0.7731905225576112 (0.0454171362817325) with: {'batch_size': 20, 'epochs': 100}
0.7680298604349236 (0.0440781759173646 7) with: {'batch_size': 40, 'epochs': 10}
0.7859136643946771 (0.0490486724321623 56) with: {'batch_size': 40, 'epochs': 50}
0.7706264199935086 (0.0544732034239734 8) with: {'batch_size': 40, 'epochs': 100}
```

드롭아웃 정규화

- 과적합을 줄이고자 하는 전략
  - ✓ 주기적으로 일부 뉴런을 제거해 나머지 뉴런들이 그 부분들을 채우게 됨
  - ✓ 어떤 뉴런이 전체 신경망에서 차지하는 중요도가 너무 올라가는 것을 방지함
- Adam 최적화 함수에서 학습률 값을 하나의 변수로 대체
  - ✓ 신경망의 매개변수를 업데이트하는 속도를 조절
- create_model() 함수에 learning_rate, dropout_rate 인자를 추가

```
seed = 6
np.random.seed(seed)

def create_model(learn_rate, dropout_rate):
  model = keras.models.Sequential()
  model.add(keras.layers.Dense(8, input_dim = 8, kernel_initializer='normal', activation='relu'))
  model.add(keras.layers.Dropout(dropout_rate))
  model.add(keras.layers.Dense(4, kernel_initializer='normal', activation='relu'))
  model.add(keras.layers.Dropout(dropout_rate))
  model.add(keras.layers.Dense(1, activation='sigmoid'))
  adam = keras.optimizers.Adam(learning_rate=learn_rate)

  model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])
  return model
```

```
model = KerasClassifier(model = create_model, epochs = 50,
                        batch_size = 40, verbose = 0)
```

```
learn_rate = [0.001, 0.01, 0.1]
dropout_rate = [0.0, 0.1, 0.2]

param_grid = dict(model__learn_rate=learn_rate, model__dropout_rate=dropout_rate)

grid = GridSearchCV(estimator = model, param_grid = param_grid,
                    cv = KFold(random_state=seed, shuffle=True),verbose = 0)
grid_results = grid.fit(X_standardized, Y)
```

드롭아웃 정규화

- 결과를 보고하는 코드
  - ✓ 테스트 데이터에 대해 78.83%의 정확도를 보임 (78.59->78.83↑)

```
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))
means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
Best: 0.7882505679974035, using {'model__dropout_rate': 0.2, 'model__learn_rate': 0.01}
0.7730606945796819 (0.028598756460473506) with: {'model__dropout_rate': 0.0, 'model__learn_rate': 0.001}
0.770399221032 1324 (0.0378626387119945) with: {'model__dropout_rate': 0.0, 'model__learn_rate': 0.01}
0.7577734501785134 (0.0280343896162 1694) with: {'model__dropout_rate': 0.0, 'model__learn_rate': 0.1}
0.7680623174294061 (0.046111143853923 36) with: {'model__dropout_rate': 0.1, 'model__learn_rate': 0.001}
0.7729957805907173 (0.026828685416583172) with: {'model__dropout_rate': 0.1, 'model__learn_rate': 0.01}
0.7576111652061019 (0.0373873864997876 9) with: {'model__dropout_rate': 0.1, 'model__learn_rate': 0.1}
0.7423888347938981 (0.039549297270884 95) with: {'model__dropout_rate': 0.2, 'model__learn_rate': 0.001}
0.7882505679974035 (0.032796247125592815) with: {'model__dropout_rate': 0.2, 'model__learn_rate': 0.01}
0.770464135021097 (0.024929972163360344) with: {'model__dropout_rate': 0.2, 'model__learn_rate': 0.1}
```

최적 초매개변수 찾기

- 이전에 찾은 최적의 학습률과 드롭아웃 비율을 적용하고, 활성함수와 초깃값에 대한 매개변수를 설정
  - ✓ kernel_initializer
  - ✓ activation
- 그리드 탐색 매개변수 설정
  - ✓ activation = ['softmax', 'relu', 'tanh', 'linear']
  - ✓ init = ['uniform', 'normal', 'zero']

```
seed = 6
np.random.seed(seed)

def create_model(init, activation):
  model = keras.models.Sequential()
  model.add(keras.layers.Dense(8, input_dim = 8, kernel_initializer=init, activation=activation))
  model.add(keras.layers.Dropout(0.2))
  model.add(keras.layers.Dense(4, kernel_initializer=init, activation=activation))
  model.add(keras.layers.Dropout(0.2))
  model.add(keras.layers.Dense(1, activation='sigmoid'))
  adam = keras.optimizers.Adam(learning_rate=0.01)

  model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])
  return model
```

```
model = KerasClassifier(model = create_model, epochs = 50,
                        batch_size = 40, verbose = 0)
```

```
activation = ['softmax', 'relu', 'tanh', 'linear']
init = ['uniform', 'normal', 'zero']

param_grid = dict(model__activation = activation, model__init = init)

grid = GridSearchCV(estimator = model, param_grid = param_grid,
                    cv = KFold(random_state=seed, shuffle=True),verbose = 0)
grid_results = grid.fit(X_standardized, Y)
```

최적 초매개변수 찾기

- 결과를 보고하는 코드
  - ✓ 테스트 데이터에 대해 79.09%의 정확도를 보임 (78.59->78.83->79.09↑)

```
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))
means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
Best: 0.7909444985394353, using {'model__activation': 'linear', 'model__init': 'uniform'}
0.7551768906199285 (0.04310156886487648) with: {'model__activation': 'softmax', 'model__init': 'uniform'}
0.7628042843232716 (0.04960811385070853) with: {'model__activation': 'softmax', 'model__init': 'normal'}
0.7653034728984096 (0.0384006710950817 5) with: {'model__activation': 'softmax', 'model__init': 'zero'}
0.7679974034404413 (0.04709672069720052) with: {'model__activation': 'relu', 'model__init': 'uniform'}
0.7832521908471277 (0.04538105351737658) with: {'model__activation': 'relu', 'model__init': 'normal'}
0.6685167153521585 (0.02715733189031009) with: {'model__activation': 'relu', 'model__init': 'zero'}
0.7679324894514767 (0.03995836996573496) with: {'model__activation': 'tanh', 'model__init': 'uniform'}
0.7831548198636806 (0.0385684705054551) with: {'model__activation': 'tanh', 'model__init': 'normal'}
0.6685167153521585 (0.02715733189031009) with: {'model__activation': 'tanh', 'model__init': 'zero'}
0.7909444985394353 (0.03832556762199145 6) with: {'model__activation': 'linear', 'model__init': 'uniform'}
0.7807854592664718 (0.03335315875606232) with: {'model__activation': 'linear', 'model__init': 'normal'}
0.6685167153521585 (0.02715733189031009) with: {'model__activation': 'linear', 'model__init': 'zero'}
```

뉴런의 개수 최적화

- 이전에 찾은 최적의 초매개변수를 적용하고, 레이어의 뉴런 개수에 대한 최적화를 시도함
  - ✓ neuron1
  - ✓ neuron2
- 그리드 탐색 매개변수 설정
  - ✓ neuron1 = [4, 8, 16]
  - ✓ neuron2 = [2, 4, 8]

```
seed = 6
np.random.seed(seed)

def create_model(neuron1, neuron2):
  model = keras.models.Sequential()
  model.add(keras.layers.Dense(neuron1, input_dim = 8, kernel_initializer='uniform', activation='linear'))
  model.add(keras.layers.Dropout(0.2))
  model.add(keras.layers.Dense(neuron2, kernel_initializer='uniform', activation='linear'))
  model.add(keras.layers.Dropout(0.2))
  model.add(keras.layers.Dense(1, activation='sigmoid'))
  adam = keras.optimizers.Adam(learning_rate=0.01)

  model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])
  return model
```

```
model = KerasClassifier(model = create_model, epochs = 50,
                        batch_size = 40, verbose = 0)
```

```
neuron1 = [4, 8, 16]
neuron2 = [2, 4, 8]

param_grid = dict(model__neuron1 = neuron1, model__neuron2 = neuron2)

grid = GridSearchCV(estimator = model, param_grid = param_grid,
                    cv = KFold(random_state=seed, shuffle=True),verbose = 0)
grid_results = grid.fit(X_standardized, Y)
```

뉴런의 개수 최적화

- 결과를 보고하는 코드
  - ✓ 테스트 데이터에 대해 79.35%의 정확도를 보임 (78.59->78.83->79.09->79.35↑)

```
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))
means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
Best: 0.7935086011035378, using {'model__neuron1': 16, 'model__neuron2': 8}
0.7833171048360921 (0.03207194515510668) with: {'model__neuron1': 4, 'model__neuron2': 2}
0.7935086011035377 (0.03630494379281224) with: {'model__neuron1': 4, 'model__neuron2': 4}
0.7832846478416098 (0.03328030756314274) with: {'model__neuron1': 4, 'model__neuron2': 8}
0.788412852969814 9 (0.03464279026440206) with: {'model__neuron1': 8, 'model__neuron2': 2}
0.7807530022719895 (0.032579127562990105) with: {'model__neuron1': 8, 'model__neuron2': 4}
0.7833171048360921 (0.03406027165506637) with: {'model__neuron1': 8, 'model__neuron2': 8}
0.7832846478416098 (0.025605174629432525) with: {'model__neuron1': 16, 'model__neuron2': 2}
0.7832846478416098 (0.03328030756314274) with: {'model__neuron1': 16, 'model__neuron2': 4}
0.7935086011035378 (0.03263414815148896) with: {'model__neuron1': 16, 'model__neuron2': 8}
```

최적의 초매개변수를 사용해 예측하기

- 최적의 초매개변수를 사용하여 데이터셋의 모든 사례에 대해 당뇨병이 발생할 지 예측해 보자
  - ✓ X_standardized를 전부 예측에 사용
- 처음 다섯 개의 행을 둘러보자
  - ✓ y_pred와 Y를 비교하면?
- 분류 보고서
  - ✓ support는 각 클래스에 해당하는 사례의 개수
    - 262명이 건강, 130명이 당뇨
  - ✓ 78% 정확도

```
# 최적 초매개변수를 가지고 예측값 생성
import numpy as np
y_pred = grid.predict(X_standardized)
```

```
print(y_pred.shape)
```

```
(392,)
```

```
print(y_pred[:5])
```

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

```
print(Y[:5])
```

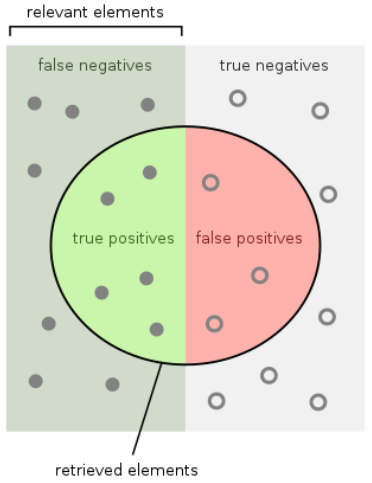
```
[0 1 1 1 1]
```

```
from sklearn.metrics import classification_report, accuracy_score
print(accuracy_score(Y, y_pred))
print(classification_report(Y, y_pred))
```

```
0.7806122448979592
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       262
           1       0.72      0.55      0.62       130

    accuracy                           0.78       392
   macro avg       0.76      0.72      0.73       392
weighted avg       0.77      0.78      0.77       392
```



https://en.wikipedia.org/wiki/Precision_and_recall

하나의 사례에 대한 예측

- 다음 코드로 하나의 사례를 뽑았을 때, 최적화된 신경망으로 이 환자가 당뇨를 가질 것인지 아닌지를 예측해 보자
- Inclass Assignment
  - ✓ 출력 결과를 확인하고 화면을 캡처하여 업로드한다

```
example = df.iloc[1]
print(example)
```

```
n_pregnant                   0.000
glucose_concentration      137.000
blood_pressure (mm Hg)      40.000
skin_thickness (mm)         35.000
serum_insulin (mu U/ml)    168.000
BMI                         43.100
pedigree_function            2.288
age                         33.000
class                        1.000
Name: 4, dtype: float64
```

```
prediction = grid.predict(X_standardized[1].reshape(1, -1))
print(prediction)
```

요약

- 케라스를 사용해 신경망을 만들고, 사이킷런 그리드 탐색법을 사용해 최적의 초매개변수를 찾는다. 초매개변수를 조정해 신경망을 최적화 하는 방법을 배웠다
- 신경망과 초매개변수 최적화 방법을 배웠고, 이를 신경망에 적용해 당뇨병 환자 데이터에서 당뇨병의 발병을 예측하는 데 응용할 수 있었다