4주. Clustering, KNN			
학번	32153180	이름	이상민

BostonHousing 데이터셋은 보스턴 지역의 지역정보 및 평균주택 가격 (medv) 정보를 담고 있다.

```
# 과제에 필요한 package
import pandas as pd
import numpy as np

from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

BostonHousing dataset에 대해 clustring을 실시하려고 한다.

Q1 Bostonhousing dataset에서 indus, dis, mdev 3개 변수(컬럼에 대한 데이터를 추출하고, 추출된 데이터에 대해 scaling을 하여 새로운 데이터셋 BH 를 생성하시오. (BH 의 앞 5개 행을 출력한다)

Source code :

```
boston = pd.read_csv('C:/Users/sangmin/Desktop/학교생활/4-2/딥러닝클라우드/dataset/BostonHousing.csv')
boston_X = boston[['indus', 'dis', 'medv']]
scaler = StandardScaler()
scaler.fit(boston_X)
BH = scaler.transform(boston_X)
BH[:5]
```

실행화면 캡쳐:

```
Out[291]:
array([[-1.2879095 ,  0.1402136 ,  0.15968566],
       [-0.59338101,  0.55715988, -0.10152429],
       [-0.59338101,  0.55715988,  1.32424667],
       [-1.30687771,  1.07773662,  1.18275795],
       [-1.30687771,  1.07773662,  1.48750288]])
```

Q2. BH 에 대해 KMeans 클러스터링을 실시하되 1~500행에 대해서만 실시하고, 클러스터의 개수는 5, random_state 의 값은 123 으로 하시오. 그리고 생성된 클러스터 값을 BH 에 추가하여 결과를 보이시오. (앞에서 10개의 행에 대해서만 결과를 보인다.)

Source code :

```
kmeans = KMeans(n_clusters=5, random_state=123).fit(BH[:500])
result = np.hstack((BH[:500], kmeans.labels_.reshape(-1, 1)))
result[:10]
```

실행화면 캡쳐:

```
Out[292]:
array([[-1.2879095 ,  0.1402136 ,  0.15968566,  2.        ],
       [-0.59338101,  0.55715988, -0.10152429,  2.        ],
       [-0.59338101,  0.55715988,  1.32424667,  1.        ],
       [-1.30687771,  1.07773662,  1.18275795,  1.        ],
       [-1.30687771,  1.07773662,  1.48750288,  1.        ],
       [-1.30687771,  1.07773662,  0.6712218 ,  4.        ],
       [-0.47665354,  0.83924392,  0.03996443,  2.        ],
       [-0.47665354,  1.02463789,  0.49708184,  4.        ],
       [-0.47665354,  1.08719646, -0.65659542,  2.        ],
       [-0.47665354,  1.32963473, -0.39538548,  4.        ]])
```

Q3. 각 클러스터의 중심점 값을 출력 하시오

Source code :

```
for i in range(5):
    print(i, '클러스터의 중심점 :', kmeans.cluster_centers_[[i]])
```

실행화면 캡쳐:

```
0 클러스터의 중심점 : [[ 1.17486908 -0.83795852 -0.7256677 ]]
1 클러스터의 중심점 : [[-1.04746396  0.03299445  1.61437165]]
2 클러스터의 중심점 : [[-0.40576204  0.08002824 -0.12606471]]
3 클러스터의 중심점 : [[ 1.12397199 -1.0298077   2.89477147]]
4 클러스터의 중심점 : [[-1.01177187  1.71875715  0.16656619]]
```

Q4. BH데이터에서 501행 이후에 대해 클러스터를 예측하여 보이시오 (데이터 + 클러스터 값을 함께 보임)

Source code :

```
pred = kmeans.predict(BH[501:])
np.hstack((BH[501:], pred.reshape(-1, 1)))
```

실행화면 캡쳐:

```
Out[295]:
array([[ 0.11573841, -0.62579623, -0.01445431,  2.        ],
       [ 0.11573841, -0.71663927, -0.21036176,  2.        ],
       [ 0.11573841, -0.77368357,  0.14880191,  2.        ],
       [ 0.11573841, -0.66843684, -0.0579893 ,  2.        ],
       [ 0.11573841, -0.61324648, -1.15724782,  0.        ]])
```

Q5. (2점) 각 클러스터별로 ndus, dis, mdev 의 평균값을 구하되 scaling 이전의 값으로 계산하여 보이시오. (1~500행을 대상으로 계산한다)

Source code :

```
kmeans = KMeans(n_clusters=5, random_state=123).fit(boston_X[:500])
result = np.hstack((boston_X[:500], kmeans.labels_.reshape(-1, 1)))
result_df = pd.DataFrame(result)


for i in range(5):
    cluster = result_df[result_df[3] == i]
    print('========', i, 'cluster ========')
    print('mean of indus : {:.2f}'.format(cluster[0].mean()))
    print('mean of dis : {:.2f}'.format(cluster[1].mean()))
    print('mean of medv : {:.2f}\n'.format(cluster[2].mean()))
```

실행화면 캡쳐:

```
======== 0 cluster ========
mean of indus : 3.68
mean of dis : 4.88
mean of medv : 32.25

======== 1 cluster ========
mean of indus : 18.81
mean of dis : 2.60
mean of medv : 21.16

======== 2 cluster ========
mean of indus : 18.93
mean of dis : 1.81
mean of medv : 12.10

======== 3 cluster ========
mean of indus : 6.95
mean of dis : 5.02
mean of medv : 21.08

======== 4 cluster ========
mean of indus : 8.71
mean of dis : 3.37
mean of medv : 47.45
```

PimaIndiansDiabetes dataset을 가지고 Classification 을 하고자 한다. (마지막의 diabetes 컬럼이 class label 임)

Q6. 데이터셋을 scaling 한 후 (diabetes 컬럼 제외) train/test set 으로 나누시오.
(test set을 30% 로 한다. random_state 는 123)
KNN 으로 분류 모델을 만드시오 (K=5)

Source code :

```
pima = pd.read_csv('C:/Users/sangmin/Desktop/학교생활/4-2/딥러닝클라우드
/dataset/PimaIndiansDiabetes.csv')
pima_X = pima.iloc[:, :-1]
pima_y = pima.iloc[:, -1]
scaler = StandardScaler()
scaler.fit(pima_X)
pima_X = pd.DataFrame(scaler.transform(pima_X))
pima_X.columns = ['pregnant', 'glucose', 'pressure', 'triceps',
'insulin', 'mass', 'pedigree', 'age']

train_X, test_X, train_y, test_y = \
    train_test_split(pima_X, pima_y, test_size=0.3 \
, random_state=123)

model = KNeighborsClassifier(n_neighbors=5)
model.fit(train_X, train_y)
```

Q7. 다음의 모델 성능 평가값을 보이시오
- training accuracy
- test accuracy
- f1 score (test set에 대해)
- precision (test set에 대해)
- recall (test set에 대해)

Source code :

```
train_pred = model.predict(train_X)
test_pred = model.predict(test_X)
print('training accuracy : {:.3f}'.format(accuracy_score(train_y, train_pred)))
print('test accuracy : {:.3f}'.format(accuracy_score(test_y, test_pred)))
print('f1 score : {:.3f}'.format(f1_score(test_y, test_pred, pos_label="pos")))
print('precision : {:.3f}'.format(precision_score(test_y, test_pred, pos_label="pos")))
print('recall : {:.3f}'.format(recall_score(test_y, test_pred, pos_label="pos")))
```

실행화면 캡쳐:

```
training accuracy : 0.816
test accuracy : 0.740
f1 score : 0.625
precision : 0.694
recall : 0.568
```

Q8. (2점) K 값을 1~10 으로 바꾸어 가면서 테스트하여 가장 높은 test accuracy 값을 도출하는 K값을 찾으시오

Source code :

```
max_acc = 0
max_K = 0

for i in range(1, 11):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(train_X, train_y)

    test_pred = model.predict(test_X)
    acc = accuracy_score(test_y, test_pred)
    print('K = {}, accuracy = {:.4f}'.format(i, acc))

    if acc > max_acc:
        max_acc = acc
        max_K = i

print('가장 높은 accuracy 값을 도출하는 K :', max_K)
```

실행화면 캡쳐:

```
K = 1, accuracy = 0.7229
K = 2, accuracy = 0.7100
K = 3, accuracy = 0.7619
K = 4, accuracy = 0.7316
K = 5, accuracy = 0.7403
K = 6, accuracy = 0.7186
K = 7, accuracy = 0.7316
K = 8, accuracy = 0.7100
K = 9, accuracy = 0.7316
K = 10, accuracy = 0.7229
가장 높은 accuracy 값을 도출하는 K : 3
```

Q9. **PimaIndiansDiabetes** 데이터셋에 대해 KNN (K=5) 으로 분류모델을 만들되 10-fold cross validation 으로 성능을 평가하시오
* random_state 는 123
- 각 fold 별 accuracy를 보이시오
- 전체 평균 accuracy를 보이시오

Source code :

```
pima_X = np.array(pima_X)
pima_y = np.array(pima_y)

kf = KFold(n_splits=10, random_state=123, shuffle=True)
model = KNeighborsClassifier(n_neighbors=5)

acc = np.zeros(10)
i = 0
for train_index, test_index in kf.split(pima_X):
    train_X, test_X = pima_X[train_index], pima_X[test_index]
    train_y, test_y = pima_y[train_index], pima_y[test_index]

    model.fit(train_X, train_y)

    pred = model.predict(test_X)

    acc[i] = accuracy_score(test_y, pred)
    print('{}-fold accuracy : {:.3f}'.format(i, acc[i]))
    i += 1

print('mean of accuracy : {:.3f}'.format(np.mean(acc)))
```

실행화면 캡쳐:

```
0-fold accuracy : 0.779
1-fold accuracy : 0.779
2-fold accuracy : 0.766
3-fold accuracy : 0.675
4-fold accuracy : 0.662
5-fold accuracy : 0.727
6-fold accuracy : 0.714
7-fold accuracy : 0.792
8-fold accuracy : 0.697
9-fold accuracy : 0.776
mean of accuracy : 0.737
```

Q10. (3점) K 값을 1~10 으로 바꾸어 가면서 테스트하여 가장 높은 test accuracy 값을 도출하는 K값을 찾으시오. 단 10-fold cross validation 으로 각 K 의 accuracy를 평가한다.
* random_state 는 123

```
max_acc = 0
max_K = 0

for i in range(1, 11):
    kf = KFold(n_splits=10, random_state=123, shuffle=True)
    model = KNeighborsClassifier(n_neighbors=i)

    print('========== K =', i, '==========')

    acc = np.zeros(10)
    i = 0
    for train_index, test_index in kf.split(pima_X):
        train_X, test_X = pima_X[train_index], pima_X[test_index]
        train_y, test_y = pima_y[train_index], pima_y[test_index]

        model.fit(train_X, train_y)

        pred = model.predict(test_X)

        acc[i] = accuracy_score(test_y, pred)

        if acc[i] > max_acc:
            max_acc = acc[i]
            max_K = i

        print('{}-fold accuracy : {:.3f}'.format(i, acc[i]))
        i += 1
    print('\n')

print('max accuracy : {:.3f} and that K : {}'.format(max_acc, max_K))
```

실행화면 캡쳐:

```
========== K = 1 ==========
0-fold accuracy : 0.727
1-fold accuracy : 0.675
2-fold accuracy : 0.740
3-fold accuracy : 0.701
4-fold accuracy : 0.701
5-fold accuracy : 0.766
6-fold accuracy : 0.766
7-fold accuracy : 0.727
8-fold accuracy : 0.658
9-fold accuracy : 0.750


========== K = 2 ==========
0-fold accuracy : 0.727
1-fold accuracy : 0.714
2-fold accuracy : 0.740
3-fold accuracy : 0.701
4-fold accuracy : 0.675
5-fold accuracy : 0.688
6-fold accuracy : 0.753
7-fold accuracy : 0.753
8-fold accuracy : 0.632
9-fold accuracy : 0.737
```

…

```
========== K = 9 ==========
0-fold accuracy : 0.753
1-fold accuracy : 0.753
2-fold accuracy : 0.701
3-fold accuracy : 0.740
4-fold accuracy : 0.727
5-fold accuracy : 0.701
6-fold accuracy : 0.701
7-fold accuracy : 0.831
8-fold accuracy : 0.737
9-fold accuracy : 0.816


========== K = 10 ==========
0-fold accuracy : 0.753
1-fold accuracy : 0.740
2-fold accuracy : 0.675
3-fold accuracy : 0.727
4-fold accuracy : 0.740
5-fold accuracy : 0.727
6-fold accuracy : 0.740
7-fold accuracy : 0.844
8-fold accuracy : 0.737
9-fold accuracy : 0.803


max accuracy : 0.844 and that K : 7
```

Q11. K-fold cross validation을 사용하는 이유를 설명하시오

고정된 training data와 test data로 모델을 만드는 경우 test data에 과적합될 가능성이 높다. 이는 처음에 나눈 test data가 데이터 중 일부분으로 고정되어 있기 때문이다. 따라서 데이터의 모든 부분을 사용해 모델을 검증하기 위해 즉, test data를 하나로 고정하지 않는 방법으로 K-fold cross validation을 사용한다.