AI스터디

주제 : 당뇨병 예측 모델 만들기

당뇨병 예측 모델 방향성

https://www.kaggle.com/code/pouryaayria/a-complete-ml-pipeline-tutorial-acu-86/notebook

머신러닝 파이프라인 주요 단계

1. 문제정의 : 혈액 성분 분석 하지 않으면서 당뇨병 예측하고 싶어
2. 데이터수집 : *Pima Indians Diabetes Database(Kaggle)*
3. 데이터준비 : 🔗 https://www.kaggle.com/uciml/pima-indians-diabetes-database
   데이터 구성 (768행 × 9열), CSV

컬럼명	설명
Pregnancies	임신 횟수
Glucose	혈당 수치
BloodPressure	혈압(mm Hg)
SkinThickness	피부 두께(mm)
Insulin	인슐린 수치
BMI	체질량지수
DiabetesPedigreeFunction	가족력 지수
Age	나이
Outcome	1=당뇨병, 0=정상 (타깃)

*부족한 데이터가 있나?

*결측치 있나? → 있으면 0 OR NA
*중복된 데이터가 있나? → 있다면 제거

*정규화 ? → 데이터의 범주별로 숫자부여(데이터 크기 축소)
*다른 유형의 정리나 매핑 수행

*완전한 특성 추출? ??

1. 데이터분리 : 훈련/(검증/)테스트 데이터셋분리
   →테스트 데이터(20%)의 변수 따로 만들어 담아야 해
2. 모델훈련 :
3. 모델평가 :Accuracy 분석(정확도 분석)
4. 모델배포 :
5. 성능모니터링 :
   소프트웨어 측
   모델 성능 모니터링 (정확도, 편향, 분산 등)
   모델 재학습 필요성
   새로운 데이터에 반영하여 모델이 주기적으로 재학습

   하드웨어 측
   cpu,메모리 등 시스템 자원 사용량이 적절한지 확인, 필요에 따라 시스템환경 조

start

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
6	148	72	35	0	33.6	0.627
1	85	66	29	0	26.6	0.351
8	183	64	0	0	23.3	0.672
1	89	66	23	94	28.1	0.167
0	137	40	35	168	43.1	2.288
5	116	74	0	0	25.6	0.201
3	78	50	32	88	31	0.248
10	115	0	0	0	35.3	0.134

csv 파일을 보았을 때 insulin, glucose, skin,bp, bmi, preg 0 값이 있음

여기서 0이 될 수 없는 항목:skin, bp, bmi

0될 수 있는거 : preg(임신)

혈액수치:glucose, insulin → 이건 제외할 생각임

📌 train_input, train_target, test_input, test_target 나눈다~
train_input : Pregnancies, BloodPressure, SkinThickness, BMI, DiabetesPedigreeFunction
train_target : outcome
test_input:glucose, insulin
test_target:outcome 하고 싶은데 이게 가능할까?

가능하지 않겠다... glucose, insulin에 대한 학습,검증 없이 다른걸로 훈련해서 glucose, insulin으로 평가한다?
이건 말이 안되고... 그럼 glucose, insulin 우선 제외하고 Pregnancies, BloodPressure, SkinThickness, BMI, DiabetesPedigreeFunction
(5) 요소로 검증해야겠다.

그럼 제일 먼저 생각해야하는거?
결측치 있나? 없음.
0값이나 na 있나?
0값 있음 : Pregnancies, BloodPressure, SkinThickness, BMI
이 중 0이 되도 되는 요소는 Pregnancies,
나머지는 0이 될 수 없어(상식적으로)
BloodPressure, SkinThickness, BMI 의 0 값을 어떻게 해볼까?

1. 단순 중앙값으로 대체
2. 평균대체
3. knn기반 결측 대체
4. 아예 제거

그리고 예측 모델을 만들기 위한 학습 방법은 뭐하는게 좋을까?
우선 outcome 이라는 답이 있으니 지도학습이 좋겠지?

1.logistic regression : 직선으로 경계를 긋고 그 경계를 기준으로 확률을 계산해서 분류한다.
→ 5개의 요소에 가중치를 곱해 값을 만든 뒤 시그모이드에 넣어
0~1 사이 확률로 바꾼다

2.knn classifier :가까운 이웃끼리 비율로 예측
스케일링이 중요한데 거리를 기준으로 가까운 이웃을 찾기 때문에
standardscaler로 꼭 평균0,분산1로 맞춘 뒤 knn 해야해!

그리고 평가는 어떻게 하지? 뭐로 잘 예측했다고 판단할 수 있지?
1.정확도(accuracy)로 평가하자
2.ROC-AUC는 “임의의 양성 샘플 하나와 음성 샘플 하나를 뽑았을 때, 모델이 양성 샘플에 더 높은 점수를 줄 확률”로 해석할 수 있다.
그래서 클래스 불균형이 있어도 전체 임계값 구간에서의 순위 성능을 잘 요약해 주기 때문에, 당뇨 예측 같은 의료 데이터 평가에 자주 사용된다.

1번째 시도

1. BloodPressure, SkinThickness, BMI의 0값 → 중앙값대체

```
cols=["BloodPressure", "SkinThickness", "BMI"]

for c in cols:
# 1) 0이 아닌 값들만 사용해서 중앙값 계산
    median_val = df.loc[df[c] != 0, c].median()

    # 2) 해당 컬럼에서 값이 0인 곳만 중앙값으로 대체
```

```
    df.loc[df[c] == 0, c] = median_val

print(df[cols].describe())
```

2. train_test_split으로 train, test 각각 나눔

```
from sklearn.model_selection import train_test_split
feature_cols=[ "Pregnancies",
    "BloodPressure",
    "SkinThickness",
    "BMI",
    "DiabetesPedigreeFunction"]
x=df[feature_cols]
y=df["Outcome"]

train_input,test_input, train_target, test_target = train_test_split(
    x,y,test_size=0.2,random_state=42
)
print(train_input.head())
print(train_target.head())
```

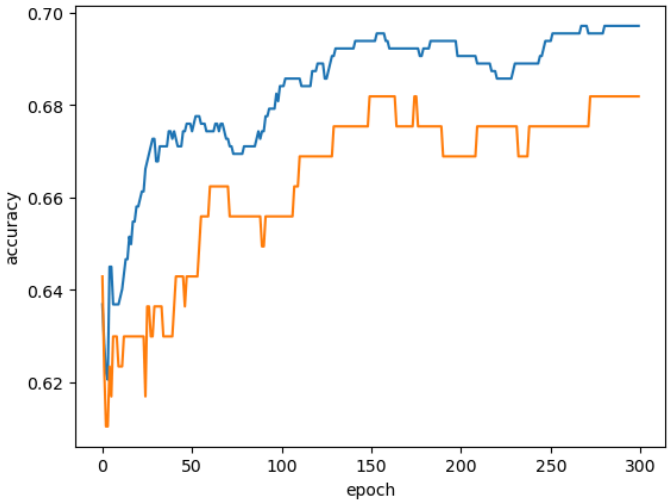
3. 4-2장의 확률적 경사 하강법을 이용해 테스트해봤어

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)

from sklearn.linear_model import SGDClassifier
#표준화 한 input 값 준비했다. 그리고 이걸 SGDClassifier로 확률경사하강법 불러온다
sc=SGDClassifier(loss='log_loss', max_iter=10, random_state=42)
#SGDClassifier 객체 만들때는 2개의 매개변수를 지정한다.
#함수로 로지스틱 손실함수를 지정 -> loss=log_loss, 에포크는 max_iter로 10 지정! 그리고 균일한 랜덤을 위해 random_state 설정!
#SGDClassifier 객체 만들때는 함수지정, 에포크 지정 해야한다!
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
0.5814332247557004
0.6038961038961039

sc.partial_fit(train_scaled, train_target)
#partial_fit은 "이어 하던 공부를 계속하는 학습” 메서드다. 한 번에 전부 처음부터 학습하는 fit과 달리, 현재까지의 가중치를 유지한 채 주어진 배치로만 추가 업데이트한다.
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
#이 코드를 실행할수록 점수가 점점 높아진다! 에포크가 늘어날수록 점수는 향상된다.
0.6384364820846905
0.6233766233766234
```

```
import numpy as np
sc = SGDClassifier(loss='log_loss', random_state=42)
train_score=[]
test_score=[]
classes = np.unique(train_target)
#train_score, test_score 결과를 담을 각각의 리스트를 준비했다.
```

```
for _ in range(0,300) :
    sc.partial_fit(train_scaled, train_target, classes=classes)
    train_score.append(sc.score(train_scaled, train_target))
    test_score.append(sc.score(test_scaled, test_target))
    #300번의 에포크를 수행하면서 그때마다 train, test 점수를 리스트에 추가함
    #이걸 그래프로 그려볼까?
```

```
import matplotlib.pyplot as plt
plt.plot(train_score)
plt.plot(test_score)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
#y축 정확도! train과 test의 갭이 가장 큰 지점의 직전! 그러니까 test가 작아지기 시작하기 직전!(과대적합되면 test 정확도 떨어짐)
#여기서는 에포크 200회
```




```
sc=SGDClassifier(loss='log_loss', max_iter=200, tol=None, random_state=42)
#tol=none 에포크 200회까지 계속 하라~
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
0.7003257328990228
0.6948051948051948
```

```
sc=SGDClassifier(loss='hinge',max_iter=200, tol=None, random_state=42)
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
#hinge로 했더니 조금 내려갔네? 그럼 log_loss가 맞겠다
0.6872964169381107
0.6818181818181818
```

4. 정리 및 결론

📌 기존 요소 7개 중 혈액검사가 아닌 항목 5개를 선택함
Pregnancies, BloodPressure, SkinThickness, BMI, DiabetesPedigreeFunction
당뇨병 결과인 outcome을 target으로 하여
지도학습인 확률적 경사하강법(SGDClassifier)으로 학습함
그리고 데이터 전처리는 BloodPressure, SkinThickness, BMI의 0값을 중앙값으로 대체한 뒤 train, test로 나눔

- StandardScaler로 평균0,분산1로 맞춤
- SGDClassifier로 학습함
  - 로지스틱 손실함수 지정, 에포크 10회 함
- partial_fit으로 점진적 학습함
- epoch를 300번 수행하여 그래프 그려봄
- 최대 정확도가 0.7이 안됨...

**결론 : epoch=200 했는데
train : 0.7003257328990228
test : 0.6948051948051948
밖에 안됨..
→ 그럼 중앙값이 아니라 아예 삭제해서 볼까?

2번째 시도

1. BloodPressure, SkinThickness, BMI의 0값 삭제

```
cols = ["BloodPressure", "SkinThickness", "BMI"]

# 세 컬럼 중 하나라도 0인 행을 모두 제거
df_clean = df[(df[cols] != 0).all(axis=1)]

print(df.shape)      # 원래 크기
print(df_clean.shape)  # 0인 행 제거 후 크기
(768, 9) -> 원래 행 크기
(537, 9) -> 0제거 후 행 크기
```

2. train_test_split으로 train, test 각각 나눔

```
from sklearn.model_selection import train_test_split

# 1) 입력으로 사용할 컬럼들
feature_cols = [
    "Pregnancies",
    "BloodPressure",
    "SkinThickness",
    "BMI",
    "DiabetesPedigreeFunction"
]
target=["Outcome"]

x = df[feature_cols]       # 입력 데이터프레임
y = df[target]
# 2) 학습/테스트 분리
train_input, test_input, train_target, test_target = train_test_split(
```

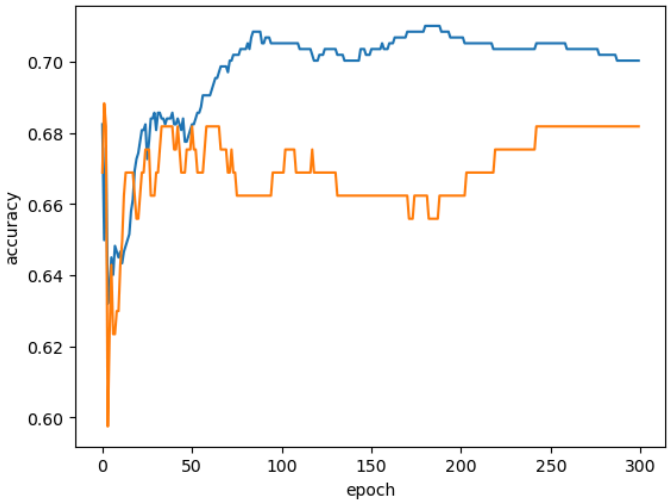
```
  x,y,
   test_size=0.2,
   random_state=42
)

print(train_input.head())
print(train_target.head())

Pregnancies  BloodPressure  SkinThickness   BMI  DiabetesPedigreeFunction
60            2            0            0   0.0                  0.304
618           9           82           24  28.2                  1.282
346           1           46           19  28.7                  0.654
294           0           50            0  21.9                  0.254
231           6           80           37  46.2                  0.238
     Outcome
60         0
618        1
346        0
294        0
231        1

→ 어? 0이 3다 있는 행(7개) 지워지지 않았네? 왜지?
```

3. 4-2장의 확률적 경사 하강법을 이용해 테스트해봤어

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
from sklearn.linear_model import SGDClassifier
#표준화 한 input 값 준비했다. 그리고 이걸 SGDClassifier로 확률경사하강법 불러온다
sc=SGDClassifier(loss='log_loss', max_iter=10, random_state=42)
#SGDClassifier 객체 만들때는 2개의 매개변수를 지정한다.
#함수로 로지스틱 손실함수를 지정 -> loss=log_loss, 에포크는 max_iter로 10 지정! 그리고 균일한 랜덤을 위해 random_state 설정!
#SGDClassifier 객체 만들때는 함수지정, 에포크 지정 해야한다!
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
0.5977198697068404
0.5909090909090909

sc.partial_fit(train_scaled, train_target)
#partial_fit은 "이어 하던 공부를 계속하는 학습” 메서드다. 한 번에 전부 처음부터 학습하는 fit과 달리, 현재까지의 가중치를 유지한 채 주어진 배치로만 추가 업데이트한다.
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
#이 코드를 실행할수록 점수가 점점 높아진다! 에포크가 늘어날수록 점수는 향상된다.
0.6465798045602605
0.6428571428571429

import numpy as np
sc = SGDClassifier(loss='log_loss', random_state=42)
train_score=[]
test_score=[]
classes = np.unique(train_target)
#train_score, test_score 결과를 담을 각각의 리스트를 준비했다.
```

```
for _ in range(0,300) :
    sc.partial_fit(train_scaled, train_target, classes=classes)
    train_score.append(sc.score(train_scaled, train_target))
    test_score.append(sc.score(test_scaled, test_target))
    #300번의 에포크를 수행하면서 그때마다 train, test 점수를 리스트에 추가함
    #이걸 그래프로 그려볼까?
```

```
import matplotlib.pyplot as plt
plt.plot(train_score)
plt.plot(test_score)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
#y축 정확도! train과 test의 갭이 가장 큰 지점의 직전! 그러니까 test가 작아지기 시작하기 직전!(과대적합되면 test 정확도 떨어짐)
#여기서는 에포크 200회
```



→근데 여전히 정확도가 0.7이 안되네?! 흠...

```
sc=SGDClassifier(loss='log_loss', max_iter=200, tol=None, random_state=42)
#tol=none 에포크 200회까지 계속 하라~
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
0.6905537459283387
0.7012987012987013
```

4. 정리 및 결론

📌 기존 요소의 BloodPressure, SkinThickness, BMI의 0값을 빼고 진행함
이후 진행은 1번째 시도와 같음

**결론 : epoch=200 했는데
train : 0.6905537459283387
test : 0.7012987012987013
1번째 시도 train : 0.7003257328990228
1번째 시도 test : 0.6948051948051948
→ 오히려 정확도가 좀 더 떨어짐 왜일까?
→다른 훈련법으로 해볼까?

3번째 시도

1. BloodPressure, SkinThickness, BMI의 0값 삭제

```
import pandas as pd

df = pd.read_csv(
    r"/mnt/c/Users/KDT-18/Desktop/AI_study/미니 프로젝트_당뇨병예측/diabetes.csv"
)

print(df.head())

cols = ["BloodPressure", "SkinThickness", "BMI"]

# 세 컬럼 중 하나라도 0인 행을 모두 제거
df_clean = df[(df[cols] != 0).all(axis=1)]

print(df.shape)      # 원래 크기
print(df_clean.shape)  # 0인 행 제거 후 크기
print(df_clean.head())
```

```
(768, 9)
(537, 9)
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1      85              66             29        0  26.6
3            1      89              66             23       94  28.1
4            0      137             40             35      168  43.1
6            3      78              50             32       88  31.0

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
3                     0.167   21        0
4                     2.288   33        1
6                     0.248   26        1
```

2. train, test 나누기~

```
from sklearn.model_selection import train_test_split

# 1) 입력으로 사용할 컬럼들
feature_cols = [
```

```
    "Pregnancies",
    "BloodPressure",
    "SkinThickness",
    "BMI",
    "DiabetesPedigreeFunction"
]
target=["Outcome"]

x = df[feature_cols]      # 입력 데이터프레임
y = df[target]
# 2) 학습/테스트 분리
train_input, test_input, train_target, test_target = train_test_split(
  x,y,
    test_size=0.2,
    random_state=42
)

print(train_input.head())
print(train_target.head())

Pregnancies  BloodPressure  SkinThickness   BMI  DiabetesPedigreeFunction
60             2             0             0   0.0                  0.304
618            9            82            24  28.2                  1.282
346            1            46            19  28.7                  0.654
294            0            50             0  21.9                  0.254
231            6            80            37  46.2                  0.238
     Outcome
60         0
618        1
346        0
294        0
231        1

→여전히 0이 3개인 행이 남아있잖아? 이걸 어쩌지? 흠...
```

3. 3-3장의 특성공학과 규제 참고해서 학습

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures()
poly.fit([[2,3]])
print(poly.transform([[2,3]]))

#2와 3을 이용해 transform → 2, 3, 2제곱, 3제곱, 2*3, 1
[[1. 2. 3. 4. 6. 9.]]

poly = PolynomialFeatures(include_bias=False)
#poly에 무조건 1이 들어가는데 우린 모델링 할 때 절편이 들어갈거라 이건 계산 안해도 된다.
#include_bias=False로 설정 -> 절편(1) 제외할게~

poly.fit([[2,3]])
print(poly.transform([[2,3]]))

poly = PolynomialFeatures(include_bias=False)
poly.fit(train_input)
train_poly = poly.transform(train_input)
print(train_poly.shape)
(614, 20)
#614개의 행에 20개의 복합특성(transform)이 추가되었어!
```

```
#어떻게 특성이 전처리 된거지? 한번 볼까?

poly.get_feature_names_out()
array(['Pregnancies', 'BloodPressure', 'SkinThickness', 'BMI',
       'DiabetesPedigreeFunction', 'Pregnancies^2',
       'Pregnancies BloodPressure', 'Pregnancies SkinThickness',
       'Pregnancies BMI', 'Pregnancies DiabetesPedigreeFunction',
       'BloodPressure^2', 'BloodPressure SkinThickness',
       'BloodPressure BMI', 'BloodPressure DiabetesPedigreeFunction',
       'SkinThickness^2', 'SkinThickness BMI',
       'SkinThickness DiabetesPedigreeFunction', 'BMI^2',
       'BMI DiabetesPedigreeFunction', 'DiabetesPedigreeFunction^2'],
      dtype=object)

#dtype이 object인걸로 봐서 문자열이네

test_poly = poly.transform(test_input)
print(test_poly.shape)
(154, 20)
```

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(train_poly, train_target)
print(lr.score(train_poly, train_target))
#0.18 나왔네!  -> 이거 별로다
print(lr.score(test_poly, test_target))
#0.16 나왔네  -> 별로네.. 다른걸로 해볼까?
0.1815793272109545
0.16148913632495443
```

4. 정리 및 결론

📌 기존 요소의 BloodPressure, SkinThickness, BMI의 0값을 빼고
3-3장의 특성공학과 규제 참고해서 학습

** 제일 최악...
train : 0.1815793272109545
test : 0.16148913632495443
1번째 시도 train : 0.7003257328990228
1번째 시도 test : 0.6948051948051948
→ 과적합이 될 줄 알았는데 전혀 아님...
→모든 요소에 대해 제곱, 곱으로 연결되는데
혹시 5개의 요소 중 진단에 영향을 주지 않는 요소가 있는걸까?
그럼 빼볼까??

4번째 시도

1. 전처리, split 동일함~

```
import pandas as pd

df = pd.read_csv(
    r"/mnt/c/Users/KDT-18/Desktop/AI_study/미니 프로젝트_당뇨병예측/diabetes.csv"
)
```

```
print(df.head())

cols = ["BloodPressure", "SkinThickness", "BMI"]

# 세 컬럼 중 하나라도 0인 행을 모두 제거
df_clean = df[(df[cols] != 0).all(axis=1)]

print(df.shape)      # 원래 크기
print(df_clean.shape)  # 0인 행 제거 후 크기
print(df_clean.head())
(768, 9)
(537, 9)

from sklearn.model_selection import train_test_split

# 1) 입력으로 사용할 컬럼들
feature_cols = [ "Pregnancies",
    "BloodPressure",
    "SkinThickness",
    "BMI",
    "DiabetesPedigreeFunction"
]
target=["Outcome"]
x = df[feature_cols]       # 입력 데이터프레임
y = df[target]
# 2) 학습/테스트 분리
train_input, test_input, train_target, test_target = train_test_split(
   x,y,
    test_size=0.2,
    random_state=42
)

print(train_input.head())
print(train_target.head())

 Pregnancies  BloodPressure  SkinThickness   BMI  DiabetesPedigreeFunction
60            2             0              0   0.0                     0.304
618           9             82            24  28.2                      1.282
346           1             46            19  28.7                     0.654
294           0             50             0  21.9                     0.254
231           6             80            37  46.2                      0.238
     Outcome
60         0
618        1
346        0
294        0
231        1
```

2. 5-3장 트리앙상블 참고해서 학습

```
import numpy as np
from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_jobs=-1, random_state=42)
scores = cross_validate(rf, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
```

1.0 0.6792216446754631

```
rf.fit(train_input, train_target)
print(rf.feature_importances_)
print(train_input.head())
```

[0.14671567 0.16093128 0.14083122 0.29626006 0.25526177]

	Pregnancies	BloodPressure	SkinThickness	BMI	DiabetesPedigreeFunction
60	2	0	0	0.0	0.304
618	9	82	24	28.2	1.282
346	1	46	19	28.7	0.654
294	0	50	0	21.9	0.254
231	6	80	37	46.2	0.238

```
rf=RandomForestClassifier(oob_score=True, n_jobs=-1, random_state=42)
rf.fit(train_input, train_target)
print(rf.oob_score_)
```

0.6612377850162866 → 검증세트의 정확도가 0.66

```
#엑스트라트리 : 부트스트랩과 다르게 중복된 샘플 없음. 한번씩만 사용 -> 무작위로 분할! 진정한 랜덤, 진정한 과대적합 방지
from sklearn.ensemble import ExtraTreesClassifier
et = ExtraTreesClassifier(n_jobs=-1, random_state=42)
scores = cross_validate(et, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
#연산 속도가 빠름!
```

1.0 0.6872984139677463

```
et.fit(train_input, train_target)
print(et.feature_importances_)
```

[0.17513467 0.18293476 0.15003651 0.26208523 0.22980882]

```
#그레디언트 부스팅: 깊이는 얕고, 경사하강법 적용 -> 결정트리를 계속 추가함
from sklearn.ensemble import GradientBoostingClassifier
gb=GradientBoostingClassifier(random_state=42)
scores = cross_validate(gb, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
```

0.8978002417498716 0.6922964147674263

```
gb=GradientBoostingClassifier(n_estimators=500, learning_rate=0.2,
                              random_state=42)
scores=cross_validate(gb, train_input, train_target,
                      return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
#결정 트리 개수를 500개, 학습률 learning rate의 기본값은 0.1 인데 0.2로 상향함
```

1.0 0.6336798613887779

```
gb.fit(train_input, train_target)
print(gb.feature_importances_)
```

```
[0.11801682 0.11354438 0.09768736 0.34479447 0.32595696]
```

```
#그레디언트 부스팅에서 속도와 성능을 개선한게 히스토그램 기반 그레디언트 부스팅
#정형 데이터의 앙상블 중 가장 인기있음.
#특성을 255+1로 나눔으로써 노드 분할 시 최적의 노드를 빨리 찾을 수 있음->평행하게 주욱~ 늘려서 최적점 찾기
#XGBoost, LightGBM 있음!
from sklearn.ensemble import HistGradientBoostingClassifier
hgb=HistGradientBoostingClassifier(random_state=42)
scores = cross_validate(hgb, train_input, train_target,
                        return_train_score=True)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
#여기서는 n_jobs 지정을 안해도 되는군...
0.9910411802692364 0.6548047447687593

#특성중요도를 permutationJmportance () 함수로 계산해야하는 번거로움이 있음!
from sklearn.inspection import permutation_importance
hgb.fit(train_input, train_target)
result=permutation_importance(hgb, train_input, train_target,
                              n_repeats=10, random_state=42, n_jobs=-1)
print(result.importances_mean)
#훈련레벨에서 중요도 체크
[0.11628664 0.11742671 0.10488599 0.25895765 0.23599349]
```

```
result = permutation_importance(hgb, test_input, test_target,
                                n_repeats=10, random_state=42, n_jobs=-1)
print(result.importances_mean)
#테스트레벨에서 중요도 체크
[0.02142857 0.00974026 0.0038961  0.07922078 0.03246753]
```

```
#최종 성능 확인하자!!
hgb.score(test_input, test_target)
0.6688311688311688
```

```
from xgboost import XGBClassifier
xgb=XGBClassifier(tree_method='hist', random_state=42)
scores=cross_validate(xgb, train_input, train_target,
                      return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
1.0 0.6320271891243504
```

3. 정리 및 결론

📌 기존 요소의 BloodPressure, SkinThickness, BMI의 0값을 빼고
5-3장의 트리앙상블 참고해서 학습

** 과적합이일어났는데 ...
train : 0.996742671009772
test : 0.6688311688311688
1번째 시도 train : 0.7003257328990228
1번째 시도 test : 0.6948051948051948
→ SkinThickness 영향을 주지 않는 요소
→BMI 중요한 요소
→ 그렇다면 SkinThickness를 빼볼까?

5번째 시도

1. 전처리 및 train, test 분리

```
import pandas as pd

df = pd.read_csv(
    r"/mnt/c/Users/KDT-18/Desktop/AI_study/미니 프로젝트_당뇨병예측/diabetes.csv"
)

print(df.head())

cols = ["BloodPressure", "BMI"]

# 세 컬럼 중 하나라도 0인 행을 모두 제거
df_clean = df[(df[cols] != 0).all(axis=1)]

print(df.shape)      # 원래 크기
print(df_clean.shape)  # 0인 행 제거 후 크기
(768, 9)
(729, 9)
```

```
from sklearn.model_selection import train_test_split

# 1) 입력으로 사용할 컬럼들
feature_cols = [ "Pregnancies",
    "BloodPressure",
    "BMI",
    "DiabetesPedigreeFunction"
]
target=["Outcome"]

x = df[feature_cols]       # 입력 데이터프레임
y = df[target]
# 2) 학습/테스트 분리
train_input, test_input, train_target, test_target = train_test_split(
   x,y,
    test_size=0.2,
    random_state=42
)

print(train_input.head())
print(train_target.head())

Pregnancies  BloodPressure   BMI  DiabetesPedigreeFunction
60            2             0   0.0                 0.304
618           9            82  28.2                 1.282
346           1            46  28.7                 0.654
294           0            50  21.9                 0.254
231           6            80  46.2                 0.238
     Outcome
60         0
618        1
346        0
```

```
294       0
231       1
```

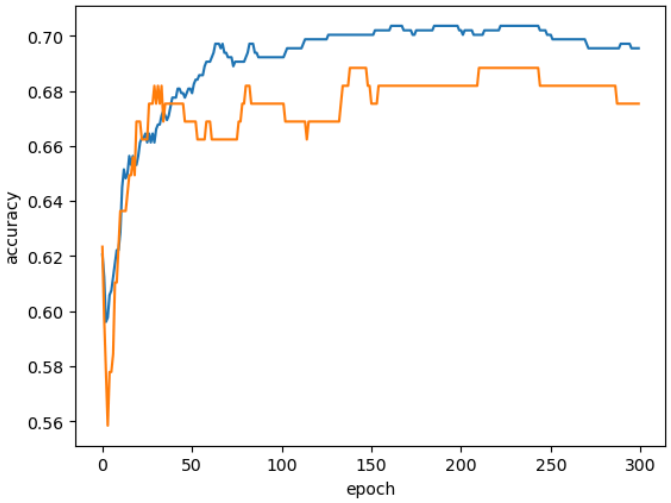
2. SGDclassifier로 다시 테스트해봄!

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
from sklearn.linear_model import SGDClassifier
#표준화 한 input 값 준비했다. 그리고 이걸 SGDClassifier로 확률경사하강법 불러온다
sc=SGDClassifier(loss='log_loss', max_iter=10, random_state=42)
#SGDClassifier 객체 만들때는 2개의 매개변수를 지정한다.
#함수로 로지스틱 손실함수를 지정 -> loss=log_loss, 에포크는 max_iter로 10 지정! 그리고 균일한 랜덤을 위해 random_state 설정!
#SGDClassifier 객체 만들때는 함수지정, 에포크 지정 해야한다!
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
0.5700325732899023
0.5844155844155844
```

```
sc.partial_fit(train_scaled, train_target)
#partial_fit은 "이어 하던 공부를 계속하는 학습" 메서드다. 한 번에 전부 처음부터 학습하는 fit과 달리, 현재까지의 가중치를 유지한 채 주어진 배치로만 추가 업데이트한다.
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
#이 코드를 실행할수록 점수가 점점 높아진다! 에포크가 늘어날수록 점수는 향상된다.
0.6286644951140065
0.6363636363636364
```

```
import numpy as np
sc = SGDClassifier(loss='log_loss', random_state=42)
train_score=[]
test_score=[]
classes = np.unique(train_target)
#train_score, test_score 결과를 담을 각각의 리스트를 준비했다.

for _ in range(0,300) :
    sc.partial_fit(train_scaled, train_target, classes=classes)
    train_score.append(sc.score(train_scaled, train_target))
    test_score.append(sc.score(test_scaled, test_target))
    #300번의 에포크를 수행하면서 그때마다 train, test 점수를 리스트에 추가함
    #이걸 그래프로 그려볼까?
```

```
import matplotlib.pyplot as plt
plt.plot(train_score)
plt.plot(test_score)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
#y축 정확도! train과 test의 갭이 가장 큰 지점의 직전! 그러니까 test가 작아지기 시작하기 직전!(과대적합되면 test 정확도 떨어짐)
#여기서는 에포크 70회
```




```
sc=SGDClassifier(loss='log_loss', max_iter=70, tol=None, random_state=42)
#tol=none 에포크 100회까지 계속 하라~
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))

0.6824104234527687
0.6688311688311688
```

```
sc=SGDClassifier(loss='hinge',max_iter=80, tol=None, random_state=42)
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))

0.6514657980456026
0.6103896103896104
```

3. 정리 및 결론

📌 기존 요소의 BloodPressure, BMI의 0값을 빼고
SkinThickness는 아예 평가 요소로 넣지 않음
첫번째 시도였던 SGDClassifier로 학습

** 0값 행 빼고 요소 빼고 한 결과보다
첫번째 중앙값이 더 결과가 좋잖아?? 흠??
train : 0.6514657980456026
test : 0.6103896103896104
4번째 시도 train : 0.996742671009772
4번째 시도 test : 0.6688311688311688
1번째 시도 train : 0.7003257328990228
1번째 시도 test : 0.6948051948051948
→ 그렇다면 SkinThickness를 빼고 랜덤 포레스트 해볼까?

6번째 시도

1.전처리 및 train, test 나누기

```
import pandas as pd

df = pd.read_csv(
    r"/mnt/c/Users/KDT-18/Desktop/AI_study/미니 프로젝트_당뇨병예측/diabetes.csv"
)

print(df.head())
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1      85              66             29        0  26.6
2            8      183             64              0        0  23.3
3            1      89              66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```
cols = ["BloodPressure", "BMI"]

# 세 컬럼 중 하나라도 0인 행을 모두 제거
df_clean = df[(df[cols] != 0).all(axis=1)]

print(df.shape)      # 원래 크기
print(df_clean.shape)  # 0인 행 제거 후 크기
print(df_clean.head())

(768, 9)
(729, 9)
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1      85              66             29        0  26.6
2            8      183             64              0        0  23.3
3            1      89              66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```
from sklearn.model_selection import train_test_split

# 1) 입력으로 사용할 컬럼들
feature_cols = [ "Pregnancies",
    "BloodPressure",
    "BMI",
    "DiabetesPedigreeFunction"
]
target=["Outcome"]
x = df[feature_cols]      # 입력 데이터프레임
```

```
y = df[target]
# 2) 학습/테스트 분리
train_input, test_input, train_target, test_target = train_test_split(
  x,y,
   test_size=0.2,
   random_state=42
)

print(train_input.head())
print(train_target.head())
```

```
Pregnancies  BloodPressure   BMI  DiabetesPedigreeFunction
60            2             0   0.0                  0.304
618           9            82  28.2                  1.282
346           1            46  28.7                  0.654
294           0            50  21.9                  0.254
231           6            80  46.2                  0.238
    Outcome
60        0
618       1
346       0
294       0
231       1
```

2. 랜덤포레스트 해보기~

```
import numpy as np
from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_jobs=-1, random_state=42)
scores = cross_validate(rf, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
```

```
1.0 0.6792483006797282
```

```
rf.fit(train_input, train_target)
print(rf.feature_importances_)
print(train_input.head())
```

```
[0.15959017 0.19412038 0.34984625 0.29644321]
    Pregnancies  BloodPressure   BMI  DiabetesPedigreeFunction
60            2             0   0.0                  0.304
618           9            82  28.2                  1.282
346           1            46  28.7                  0.654
294           0            50  21.9                  0.254
231           6            80  46.2                  0.238
```

```
rf=RandomForestClassifier(oob_score=True, n_jobs=-1, random_state=42)
rf.fit(train_input, train_target)
print(rf.oob_score_)
```

```
0.6644951140065146
```

```
#엑스트라트리 : 부트스트랩과 다르게 중복된 샘플 없음. 한번씩만 사용 -> 무작위로 분할! 진정한 랜덤, 진정한 과대적합 방지
from sklearn.ensemble import ExtraTreesClassifier
et = ExtraTreesClassifier(n_jobs=-1, random_state=42)
scores = cross_validate(et, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
```

```
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
#연산 속도가 빠름!

1.0 0.6808210049313608
```

```
et.fit(train_input, train_target)
print(et.feature_importances_)
[0.1741412  0.21948697 0.33006678 0.27630506]
```

```
#그레디언트 부스팅: 깊이는 얕고, 경사하강법 적용 -> 결정트리를 계속 추가함
from sklearn.ensemble import GradientBoostingClassifier
gb=GradientBoostingClassifier(random_state=42)
scores = cross_validate(gb, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
0.8908780818969086 0.6906570705051313
```

```
gb=GradientBoostingClassifier(n_estimators=500, learning_rate=0.2,
                              random_state=42)
scores=cross_validate(gb, train_input, train_target,
                      return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
#결정 트리 개수를 500개, 학습률 learning rate의 기본값은 0.1 인데 0.2로 상향함
1.0 0.6450753032120485
```

```
gb.fit(train_input, train_target)
print(gb.feature_importances_)
[0.13811545 0.09790514 0.36476092 0.39921849]
```

```
#그레디언트 부스팅에서 속도와 성능을 개선한게 히스토그램 기반 그레디언트 부스팅
#정형 데이터의 앙상블 중 가장 인기있음.
#특성을 255+1로 나눔으로써 노드 분할 시 최적의 노드를 빨리 찾을 수 있음->평행하게 주욱~ 늘려서 최적점 찾기
#XGBoost, LightGBM 있음!
from sklearn.ensemble import HistGradientBoostingClassifier
hgb=HistGradientBoostingClassifier(random_state=42)
scores = cross_validate(hgb, train_input, train_target,
                        return_train_score=True)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
#여기서는 n_jobs 지정을 안해도 되는군...
0.98167916811551 0.6564574170331868
```

```
#특성중요도를 permutationJmportance () 함수로 계산해야하는 번거로움이 있음!
from sklearn.inspection import permutation_importance
hgb.fit(train_input, train_target)
result=permutation_importance(hgb, train_input, train_target,
                              n_repeats=10, random_state=42, n_jobs=-1)
print(result.importances_mean)
#훈련레벨에서 중요도 체크
[0.13827362 0.13843648 0.27638436 0.24527687]
```

```
result = permutation_importance(hgb, test_input, test_target,
                                n_repeats=10, random_state=42, n_jobs=-1)
print(result.importances_mean)
```

```
#테스트레벨에서 중요도 체크
[0.03441558 0.02402597 0.08961039 0.0487013 ]
```

```
#최종 성능 확인하자!!
hgb.score(test_input, test_target)
0.6818181818181818
hgb.score(train_input, train_target)
0.9788273615635179
```

```
from xgboost import XGBClassifier
xgb=XGBClassifier(tree_method='hist', random_state=42)
scores=cross_validate(xgb, train_input, train_target,
                      return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
1.0 0.6434359589497535
```

3. 정리 및 결론

📌 기존 요소의 BloodPressure, BMI의 0값을 빼고 SkinThickness를 뺀 요소 4개로 랜덤포레스트 학습하여 모델링

** 결과 비교
train : 0.9788273615635179
test : 0.6818181818181818

랜덤포레스트
4번째 시도 train : 0.996742671009772
4번째 시도 test : 0.6688311688311688
→ SkinThickness를 뺀 요소 4개로 랜덤포레스트 학습하였을 때
전보다 과대적합은 조금 줄고 테스트에서 정확도는 좀 더 올라감

SGDClassifier
1번째 시도 train : 0.7003257328990228
1번째 시도 test : 0.6948051948051948
→ SGDClassifier는 0값을 중앙값으로 둔게 제일 확률이 높았음