스마트폰 센서 데이터 기반 모션 분류

단계3 : 단계별 모델링

0.미션

단계별로 나눠서 모델링을 수행하고자 합니다.

- 단계1 : 정적(0), 동적(1) 행동 분류 모델 생성
- 단계2 : 세부 동작에 대한 분류모델 생성
  - 단계1 모델에서 0으로 예측 -> 정적 행동 3가지 분류 모델링
  - 단계1 모델에서 1으로 예측 -> 동적 행동 3가지 분류 모델링
- 모델 통합
  - 두 단계 모델을 통합하고, 새로운 데이터에 대해서 최종 예측결과와 성능평가가 나오도록 함수로 만들기
- 성능 비교
  - 기본 모델링의 성능과 비교
  - 모든 모델링은 [다양한 알고리즘 + 성능 튜닝]을 수행해야 합니다.
  - 성능 가이드
    ```
    * Accuracy : 0.980~1.00
    ```

1.환경설정

- 세부 요구사항
  - 경로 설정 : 다음의 두가지 방법 중 하나를 선택하여 폴더를 준비하고 데이터를 로딩하시오.
    - 1) 로컬 수행(Ananconda)
      - 제공된 압축파일을 다운받아 압축을 풀고
      - anaconda의 root directory(보통 C:/Users/< ID > 에 project 폴더를 만들고, 복사해 넣습니다.
    - 2) 구글콜랩
      - 구글 드라이브 바로 밑에 project 폴더를 만들고,
      - 데이터 파일을 복사해 넣습니다.
  - 기본적으로 필요한 라이브러리를 import 하도록 코드가 작성되어 있습니다.
    - 필요하다고 판단되는 라이브러리를 추가하세요.

(1) 경로 설정

1) 로컬 수행(Anaconda)

- project 폴더에 필요한 파일들을 넣고, 본 파일을 열었다면, 별도 경로 지정이 필요하지 않습니다.

In [1]:
```
path = 'C:/Users/User/Desktop/'
```

2) 구글 콜랩 수행

- 구글 드라이브 연결

In [2]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

In [3]:
```
# path = '/content/drive/MyDrive/project/'
```

(2) 라이브러리 불러오기

1) 라이브러리 로딩

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import joblib

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import *
```

2) 제공 함수 생성

- 변수 중요도를 시각화할 수 있는 함수를 제공합니다.
- 입력 :
  - importance : 트리모델의 변수 중요도(예: model.feature*importances*)
  - names : 변수 이름 목록(예 : x_train.columns
  - result_only : 변수 중요도 순으로 데이터프레임만 return할지, 그래프도 포함할지 결정. False이면 결과 데이터프레임 + 그래프
  - topn : 중요도 상위 n개만 표시. all 이면 전체.
- 출력 :
  - 중요도 그래프 : 중요도 내림차순으로 정렬
  - 중요도 데이터프레임 : 중요도 내림차순으로 정렬

In [6]:
```
# 변수의 특성 중요도 계산하기
def plot_feature_importance(importance, names, result_only = False, topn = 'all'):
    feature_importance = np.array(importance)
    feature_name = np.array(names)

    data={'feature_name':feature_name,'feature_importance':feature_importance}
    fi_temp = pd.DataFrame(data)

    #변수의 특성 중요도 순으로 정렬하기
    fi_temp.sort_values(by=['feature_importance'], ascending=False,inplace=True)
    fi_temp.reset_index(drop=True, inplace = True)

    if topn == 'all' :
        fi_df = fi_temp.copy()
    else :
        fi_df = fi_temp.iloc[:topn]

    #변수의 특성 중요도 그래프로 그리기
    if result_only == False :
        plt.figure(figsize=(10,20))
        sns.barplot(x='feature_importance', y='feature_name', data = fi_df)

        plt.xlabel('importance')
        plt.ylabel('feature name')
        plt.grid()

    return fi_df
```

(3) 데이터 불러오기

- 주어진 데이터셋
  - data01_train.csv : 학습 및 검증용
  - data01_test.csv : 테스트용
  - feature.csv : feature 이름을 계층구조로 정리한 데이터
- 세부 요구사항
  - 칼럼 삭제 : data01_train.csv와 data01_test.csv 에서 'subject' 칼럼은 불필요하므로 삭제합니다.

1) 데이터로딩

In [7]:
```
file1 = 'data01_train.csv'
file2 = 'data01_test.csv'
file3 = 'features.csv'
```

In [8]:
```
data = pd.read_csv(path + file1)
test = pd.read_csv(path + file2)
features = pd.read_csv(path + file3)
```

In [9]:
```
# 불필요한 칼럼 삭제
data.drop('subject', axis=1, inplace=True)
test.drop('subject', axis=1, inplace=True)
```

2) 기본 정보 조회

In [10]:
```
#전체 데이터의 행,열 개수 확인
data.shape
```

Out[10]:
```
(5881, 562)
```

In [11]:
```
#전체 데이터의 상위 5개 행 확인
data.head()
```

Out[11]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBody ma
0	0.288508	-0.009196	-0.103362	-0.988986	-0.962797	-0.967422	-0.989000	-0.962596	-0.96
1	0.265757	-0.016576	-0.098163	-0.989551	-0.994636	-0.987435	-0.990189	-0.993870	-0.98
2	0.278709	-0.014511	-0.108717	-0.997720	-0.981088	-0.994008	-0.997934	-0.982187	-0.99
3	0.289795	-0.035536	-0.150354	-0.231727	-0.006412	-0.338117	-0.273557	0.014245	-0.34
4	0.394807	0.034098	0.091229	0.088489	-0.106636	-0.388502	-0.010469	-0.109680	-0.34

5 rows × 562 columns

In [12]:
```
#전체 데이터의 수치형 변수 분포 확인
data.describe()
```

Out[12]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	t
count	5881.000000	5881.000000	5881.000000	5881.000000	5881.000000	5881.000000	5881.000000	58
mean	0.274811	-0.017799	-0.109396	-0.603138	-0.509815	-0.604058	-0.628151	
std	0.067614	0.039422	0.058373	0.448807	0.501815	0.417319	0.424345	
min	-0.503823	-0.684893	-1.000000	-1.000000	-0.999844	-0.999667	-1.000000	
25%	0.262919	-0.024877	-0.121051	-0.992774	-0.977680	-0.980127	-0.993602	
50%	0.277154	-0.017221	-0.108781	-0.943933	-0.844575	-0.856352	-0.948501	
75%	0.288526	-0.010920	-0.098163	-0.242130	-0.034499	-0.262690	-0.291138	
max	1.000000	1.000000	1.000000	1.000000	0.916238	1.000000	1.000000	

8 rows × 561 columns

In [13]:
```
#전체 데이터의 모든 변수 확인
data.columns
```

Out[13]:
```
Index(['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z',
       'tBodyAcc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z',
       'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y', 'tBodyAcc-mad()-Z',
       'tBodyAcc-max()-X',
       ...
       'fBodyBodyGyroJerkMag-skewness()', 'fBodyBodyGyroJerkMag-kurtosis()',
       'angle(tBodyAccMean,gravity)', 'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'Activity'],
      dtype='object', length=562)
```

2.데이터 전처리

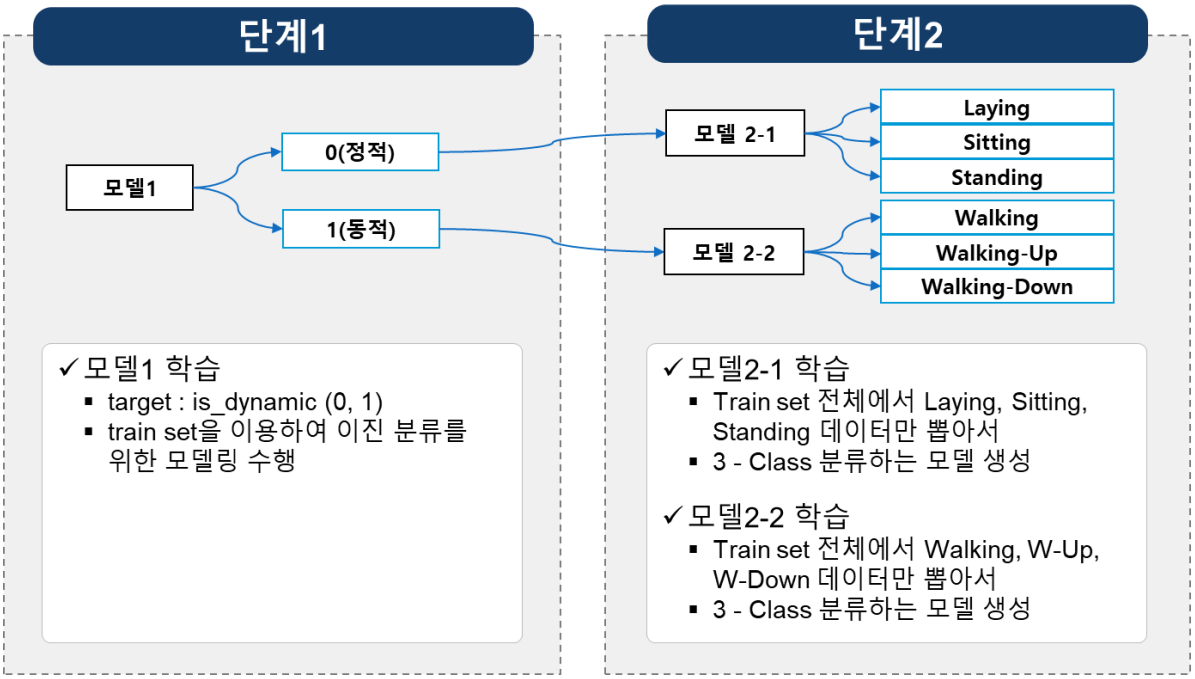
- 세부 요구사항
  - Label 추가 : data 에 Activity_dynamic 를 추가합니다. Activity_dynamic은 과제1에서 is_dynamic과 동일한 값입니다.
  - x와 y1, y2로 분할하시오.
    - y1 : Activity
    - y2 : Activity_dynamic
  - train : val = 8 : 2 혹은 7 : 3
  - random_state 옵션을 사용하여 다른 모델과 비교를 위해 성능이 재현되도록 합니다.

In [14]:
```
data['Activity_dynamic'] = data['Activity'].map({'WALKING':1, 'WALKING_UPSTAIRS':1, 'WALKING_D
                                                 'LAYING':0, 'STANDING':0,'SITTING':0})
```

In [15]:
```
x = data.drop(['Activity_dynamic',  'Activity'], axis = 1)
y1 = data.loc[:, 'Activity']
y2 = data.loc[:, 'Activity_dynamic']

x_train, x_val, y1_train, y1_val = train_test_split(x, y1, test_size = .2, random_state = 2023
x_train, x_val, y2_train, y2_val = train_test_split(x, y2, test_size = .2, random_state = 2023
```

3.단계별 모델링



(1) 단계1 : 정적/동적 행동 분류 모델

- 세부 요구사항
  - 정적 행동(Laying, Sitting, Standing)과 동적 행동(동적 : Walking, Walking-Up, Walking-Down)을 구분하는 모델 생성.
  - 몇가지 모델을 만들고 가장 성능이 좋은 모델을 선정하시오.

1) Logistic Regression

In [16]:
```
m1_1 = LogisticRegression()
m1_1.fit(x_train, y2_train)
```

Out[16]:
```
LogisticRegression
LogisticRegression()
```

In [17]:
```
p1_1 = m1_1.predict(x_val)
print('accuracy :',accuracy_score(y2_val, p1_1))
print('='*60)
print(confusion_matrix(y2_val, p1_1))
print('='*60)
print(classification_report(y2_val, p1_1))
```

```
accuracy : 0.9991503823279524
============================================================
[[656   1]
 [  0 520]]
============================================================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       657
           1       1.00      1.00      1.00       520

    accuracy                           1.00      1177
   macro avg       1.00      1.00      1.00      1177
weighted avg       1.00      1.00      1.00      1177
```

2) RandomForest

In [18]:
```
m1_2 = RandomForestClassifier()
m1_2.fit(x_train, y2_train)
```

Out[18]:
```
RandomForestClassifier()
```

In [19]:
```
p1_2 = m1_2.predict(x_val)
print('accuracy :',accuracy_score(y2_val, p1_2))
print('='*60)
print(confusion_matrix(y2_val, p1_2))
print('='*60)
print(classification_report(y2_val, p1_2))
```

```
accuracy : 0.9991503823279524
============================================================
[[656   1]
 [  0 520]]
============================================================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       657
           1       1.00      1.00      1.00       520

    accuracy                           1.00      1177
   macro avg       1.00      1.00      1.00      1177
weighted avg       1.00      1.00      1.00      1177
```

(2) 단계2-1 : 정적 동작 세부 분류

- 세부 요구사항
  - 정적 행동(Laying, Sitting, Standing)인 데이터 추출
  - Laying, Sitting, Standing 를 분류하는 모델을 생성
  - 몇가지 모델을 만들고 가장 성능이 좋은 모델을 선정하시오.

In [20]:
```
x_train2_0 = x_train[y2_train == 0]
y_train2_0 = y1_train[y2_train == 0]
```

```
x_val2_0 = x_val[y2_val == 0]
y_val2_0 = y1_val[y2_val == 0]
```

In [21]:
```
model2_0 = LogisticRegression()
model2_0.fit(x_train2_0, y_train2_0)
pred2_0 = model2_0.predict(x_val2_0)
```

```
c:\Users\User\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWa
rning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [22]:
```
print('accuracy :',accuracy_score(y_val2_0, pred2_0))
print('='*60)
print(confusion_matrix(y_val2_0, pred2_0))
print('='*60)
print(classification_report(y_val2_0, pred2_0))
```

```
accuracy : 0.9817351598173516
============================================================
[[230   0   0]
 [  0 203   6]
 [  0   6 212]]
============================================================
              precision    recall  f1-score   support

      LAYING       1.00      1.00      1.00       230
     SITTING       0.97      0.97      0.97       209
    STANDING       0.97      0.97      0.97       218

    accuracy                           0.98       657
   macro avg       0.98      0.98      0.98       657
weighted avg       0.98      0.98      0.98       657
```

(3) 단계2-2 : 동적 동작 세부 분류

- 세부 요구사항
  - 동적 행동(Walking, Walking Upstairs, Walking Downstairs)인 데이터 추출
  - Walking, Walking Upstairs, Walking Downstairs 를 분류하는 모델을 생성
  - 몇가지 모델을 만들고 가장 성능이 좋은 모델을 선정하시오.

In [23]:
```
x_train2_1 = x_train[y2_train == 1]
y_train2_1 = y1_train[y2_train == 1]

x_val2_1 = x_val[y2_val == 1]
y_val2_1 = y1_val[y2_val == 1]
```

In [24]:
```
model2_1 = LogisticRegression()
```

```
model2_1.fit(x_train2_1, y_train2_1)
pred2_1 = model2_1.predict(x_val2_1)
```

```
c:\Users\User\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWa
rning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [25]:
```
print('accuracy :',accuracy_score(y_val2_1, pred2_1))
print('='*60)
print(confusion_matrix(y_val2_1, pred2_1))
print('='*60)
print(classification_report(y_val2_1, pred2_1))
```

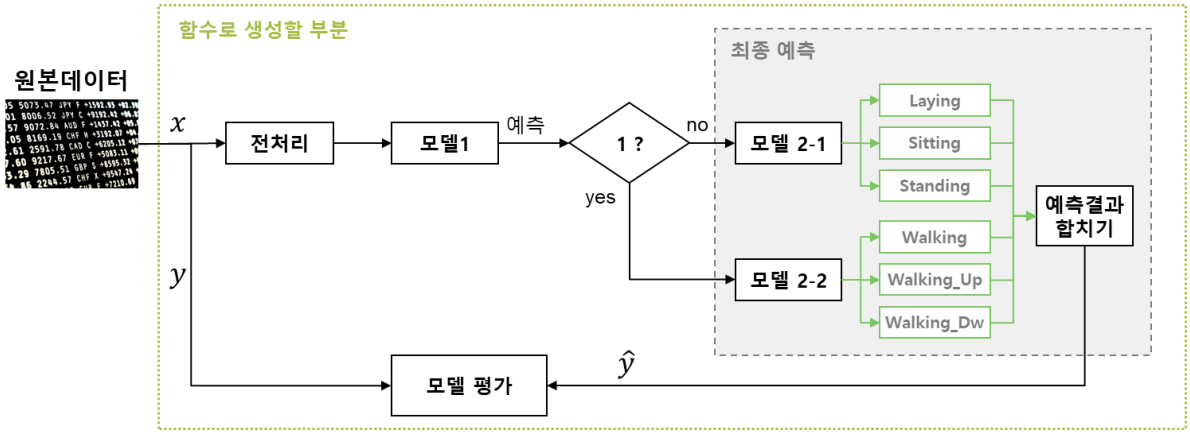
```
accuracy : 0.9961538461538462
============================================================
[[180   0   0]
 [  0 165   0]
 [  2   0 173]]
============================================================
                    precision    recall  f1-score   support

           WALKING       0.99      1.00      0.99       180
WALKING_DOWNSTAIRS       1.00      1.00      1.00       165
  WALKING_UPSTAIRS       1.00      0.99      0.99       175

          accuracy                           1.00       520
         macro avg       1.00      1.00      1.00       520
      weighted avg       1.00      1.00      1.00       520
```

(4) 분류 모델 합치기

- 세부 요구사항
  - 두 단계 모델을 통합하고, 새로운 데이터(test)에 대해서 최종 예측결과와 성능평가가 나오도록 함수로 만들기
  - 데이터 파이프라인 구축 : test데이터가 로딩되어 전처리 과정을 거치고, 예측 및 성능 평가 수행



1) 함수 만들어서 분류모델 합치기

In [26]:
```
def step_model(model1, model2_0, model2_1, data):

    # 1. 전처리

    ## 1) label 추가
    data['Activity_dynamic'] = data['Activity'].map({'WALKING':1, 'WALKING_UPSTAIRS':1, 'WALKI
                                    'LAYING':0, 'STANDING':0,'SITTING':0})

    ## 2) x, y 분할
    target = ['Activity_dynamic',  'Activity']
    x = data.drop(target, axis = 1)
    y = data.loc[:, target]

    # 2.예측

    ## 1) 단계1 모델로 0,1 구분
    pred1 = model1.predict(x)

    ## 2) 단계1의 결과로 데이터 나누기
    x_val_f2_0 = x[pred1 == 0]
    y_val_f2_0 = y[pred1 == 0]
    x_val_f2_1 = x[pred1 == 1]
    y_val_f2_1 = y[pred1 == 1]

    ## 3) 단계2 모델로 예측.
    pred2_0 = model2_0.predict(x_val_f2_0)
    pred2_1 = model2_1.predict(x_val_f2_1)

    ## 4) 하나로 합치기
    ### 예측결과
    p_f = np.r_[pred2_0, pred2_1]

    ### y도 하나로 합치기기
    y_f = np.r_[y_val_f2_0.Activity, y_val_f2_1.Activity]

    # 3.최종 성능평가
    print('accuracy :',accuracy_score(y_f, p_f))
    print('='*60)
    print(confusion_matrix(y_f, p_f))
    print('='*60)
    print(classification_report(y_f, p_f))
```

2) test 셋으로 예측하고 평가하기

In [27]:
```
test.head()
```

Out[27]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBody ma
0	0.284379	-0.021981	-0.116683	-0.992490	-0.979640	-0.963321	-0.992563	-0.977304	-0.95
1	0.277440	-0.028086	-0.118412	-0.996620	-0.927676	-0.972294	-0.997346	-0.931405	-0.97
2	0.305833	-0.041023	-0.087303	0.006880	0.182800	-0.237984	0.005642	0.028616	-0.23
3	0.276053	-0.016487	-0.108381	-0.995379	-0.983978	-0.975854	-0.995877	-0.985280	-0.97
4	0.271998	0.016904	-0.078856	-0.973468	-0.702462	-0.869450	-0.979810	-0.711601	-0.85

5 rows × 562 columns

- 성능 평가

In [28]:
```
step_model(m1_2, model2_0, model2_1, test)
```

```
accuracy : 0.9802855200543847
============================================================
[[292   0   0   0   0   0]
 [  0 240  14   0   0   0]
 [  0  13 274   0   0   0]
 [  0   0   0 228   0   0]
 [  0   0   0   0 194   1]
 [  0   0   0   1   0 214]]
============================================================
                    precision    recall  f1-score   support

           LAYING       1.00      1.00      1.00       292
          SITTING       0.95      0.94      0.95       254
         STANDING       0.95      0.95      0.95       287
          WALKING       1.00      1.00      1.00       228
WALKING_DOWNSTAIRS       1.00      0.99      1.00       195
  WALKING_UPSTAIRS       1.00      1.00      1.00       215

          accuracy                           0.98      1471
         macro avg       0.98      0.98      0.98      1471
      weighted avg       0.98      0.98      0.98      1471
```