스마트폰 센서 데이터 기반 모션 분류

단계2 : 기본 모델링

0.미션

- 데이터 전처리
  - 가변수화, 데이터 분할, NaN 확인 및 조치, 스케일링 등 필요한 전처리 수행
- 다양한 알고리즘으로 분류 모델 생성
  - 최소 4개 이상의 알고리즘을 적용하여 모델링 수행
  - 성능 비교
  - 각 모델의 성능을 저장하는 별도 데이터 프레임을 만들고 비교
- 옵션 : 다음 사항은 선택사항입니다. 시간이 허용하는 범위 내에서 수행하세요.
  - 상위 N개 변수를 선정하여 모델링 및 성능 비교
    - 모델링에 항상 모든 변수가 필요한 것은 아닙니다.
    - 변수 중요도 상위 N개를 선정하여 모델링하고 타 모델과 성능을 비교하세요.
    - 상위 N개를 선택하는 방법은, 변수를 하나씩 늘려가며 모델링 및 성능 검증을 수행하여 적절한 지점을 찾는 것입니다.
    - 성능 가이드
      - Accuracy : 0.900~0.968

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 [13]:
```
path = 'C:/Users/User/Desktop/'
```

2) 구글 콜랩 수행

- 구글 드라이브 연결

In [14]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

In [15]:
```
# path = '/content/drive/MyDrive/project/'
```

(2) 라이브러리 불러오기

1) 라이브러리 로딩

In [16]:
```
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.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
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 [17]:
```
# 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 [18]:
```
file1 = 'data01_train.csv'
file2 = 'data01_test.csv'
file3 = 'features.csv'
```

In [19]:
```
data = pd.read_csv(file1)
test = pd.read_csv(file2)
features = pd.read_csv(file3)
```

In [20]:
```
# 불필요한 칼럼 삭제
data.drop('subject', axis=1, inplace=True)
test.drop('subject', axis=1, inplace=True)
```

2) 기본 정보 조회

In [21]:
```
#전체 데이터의 행,열 개수 확인
data.shape
```

Out[21]:
```
(5881, 562)
```

In [22]:
```
#전체 데이터의 상위 5개 행 확인
data.head()
```

Out[22]:

	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
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 [23]:
```
#전체 데이터의 수치형 변수 분포 확인
data.describe()
```

Out[23]:

	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 [24]:
```
#전체 데이터의 모든 변수 확인
data.columns
```

Out[24]:
```
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

- 가변수화, 데이터 분할, NaN 확인 및 조치, 스케일링 등 필요한 전처리를 수행한다.

(1) 데이터 분할1 : x, y

- 세부 요구사항
  - x, y로 분할합니다.

In [25]:
```
target = 'Activity'
x = data.drop(target, axis = 1)
y = data.loc[:, target]
```

(2) 스케일링(필요시)

- 세부 요구사항
  - 스케일링을 필요로 하는 알고리즘 사용을 위해서 코드 수행
  - min-max 방식 혹은 standard 방식 중 한가지 사용.

In [26]:
```
scaler = MinMaxScaler()
x_s = scaler.fit_transform(x)
```

(3) 데이터분할2 : train, validation

- 세부 요구사항
  - train : val = 8 : 2 혹은 7 : 3
  - random_state 옵션을 사용하여 다른 모델과 비교를 위해 성능이 재현되도록 합니다.

In [27]:
```
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = .2, random_state = 2023)
x_train_s, x_val_s, y_train_s, y_val_s = train_test_split(x_s, y, test_size = .2, random_state
```

3. 기본 모델링

- 세부 요구사항
  - 최소 4개 이상의 알고리즘을 적용하여 모델링을 수행한다.
  - 각 알고리즘별로 전체 변수로 모델링, 상위 N개 변수를 선택하여 모델링을 수행하고 성능 비교를 한다.

(1) Random Forest Classifier

1) 전체 변수

In [28]:
```
m1_1 = RandomForestClassifier()

m1_1.fit(x_train, y_train)
p1_1 = m1_1.predict(x_val)
```

In [29]:
```
print('accuracy :',accuracy_score(y_val, p1_1))
print('='*60)
print(confusion_matrix(y_val, p1_1))
print('='*60)
print(classification_report(y_val, p1_1))
```

```
accuracy : 0.9787595581988106
============================================================
[[230   0   0   0   0   0]
 [  0 197  12   0   0   0]
 [  0   4 214   0   0   0]
 [  0   0   0 176   3   1]
 [  0   0   0   2 162   1]
 [  0   0   0   0   2 173]]
============================================================
                    precision    recall  f1-score   support

           LAYING       1.00      1.00      1.00       230
          SITTING       0.98      0.94      0.96       209
         STANDING       0.95      0.98      0.96       218
          WALKING       0.99      0.98      0.98       180
WALKING_DOWNSTAIRS       0.97      0.98      0.98       165
  WALKING_UPSTAIRS       0.99      0.99      0.99       175

          accuracy                           0.98      1177
         macro avg       0.98      0.98      0.98      1177
      weighted avg       0.98      0.98      0.98      1177
```

In [30]:
```
r = plot_feature_importance(m1_1.feature_importances_, list(x_train), True)
```

In [31]:
```
r.head()
```

Out[31]:

	feature_name	feature_importance
0	tGravityAcc-energy()-X	0.036415
1	tGravityAcc-max()-X	0.030172
2	tGravityAcc-mean()-X	0.029687
3	angle(X,gravityMean)	0.026682
4	tGravityAcc-min()-X	0.026528

2) (옵션)적절히 선택한 변수

- 변수 중요도 상위 100로 결정해서 모델링 해보기
- 변수 중요도 상위 1 ~ 400 까지 변수를 하나씩 늘려가며 모델링 및 성능 비교

In [32]:
```
feature100 = r.loc[:99, 'feature_name']
```

In [33]:
```
x_train100 = x_train[feature100]
x_val100 = x_val[feature100]
```

In [34]:
```
m1_2 = RandomForestClassifier()

m1_2.fit(x_train100, y_train)
p1_2 = m1_2.predict(x_val100)
```

In [35]:
```
print('accuracy :',accuracy_score(y_val, p1_2))
print('='*60)
print(confusion_matrix(y_val, p1_2))
print('='*60)
print(classification_report(y_val, p1_2))
```

```
accuracy : 0.9813084112149533
============================================================
[[230   0   0   0   0   0]
 [  0 202   7   0   0   0]
 [  0   3 215   0   0   0]
 [  0   0   0 174   3   3]
 [  0   0   0   2 160   3]
 [  0   0   0   0   1 174]]
============================================================
                    precision    recall  f1-score   support

           LAYING       1.00      1.00      1.00       230
          SITTING       0.99      0.97      0.98       209
         STANDING       0.97      0.99      0.98       218
          WALKING       0.99      0.97      0.98       180
WALKING_DOWNSTAIRS       0.98      0.97      0.97       165
  WALKING_UPSTAIRS       0.97      0.99      0.98       175

          accuracy                           0.98      1177
         macro avg       0.98      0.98      0.98      1177
      weighted avg       0.98      0.98      0.98      1177
```

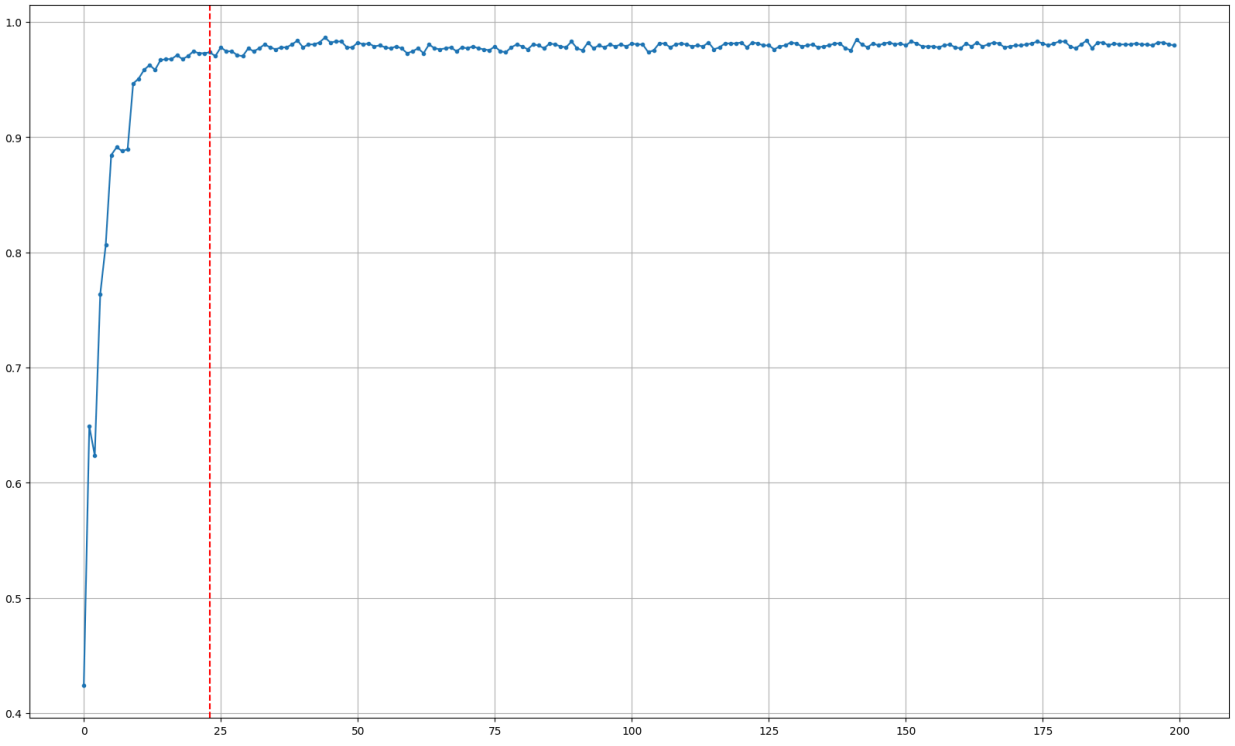
- 가장 중요한 변수부터 하나씩 증가시켜가며 모델링 및 accuracy 구하기

In [36]:
```
acc = []

for i in range(200) :   # 전체 변수는 561이지만, 200 정도면 충분
    featureN = r.loc[:i, 'feature_name']
    x_trainN = x_train[featureN]
    x_valN = x_val[featureN]
    m = RandomForestClassifier()
    m.fit(x_trainN, y_train)
    p = m.predict(x_valN)
    acc.append(accuracy_score(y_val, p))
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
```

In [56]:
```
plt.figure(figsize = (20,12))
plt.plot(range(200), acc, marker = '.')
plt.axvline(x=23, color='red', linestyle='--')
plt.grid()
plt.show()
```



- 적절한 지점의 성능에 맞춰 변수 선택 : ==> 30~50개 정도면 충분!

In [38]:
```
feature50 = r.loc[:49, 'feature_name']
x_train50 = x_train[feature50]
x_val50 = x_val[feature50]
```

In [39]:
```
m1_2 = RandomForestClassifier()
m1_2.fit(x_train50, y_train)
p1_2 = m1_2.predict(x_val50)
```

In [40]:
```
print('accuracy :',accuracy_score(y_val, p1_2))
print('='*60)
print(confusion_matrix(y_val, p1_2))
print('='*60)
print(classification_report(y_val, p1_2))
```

```
accuracy : 0.9787595581988106
============================================================
[[230   0   0   0   0   0]
 [  0 203   6   0   0   0]
 [  0   6 212   0   0   0]
 [  0   0   0 175   2   3]
 [  0   0   0   4 159   2]
 [  0   0   0   1   1 173]]
============================================================
                    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
          WALKING       0.97      0.97      0.97       180
WALKING_DOWNSTAIRS       0.98      0.96      0.97       165
  WALKING_UPSTAIRS       0.97      0.99      0.98       175

          accuracy                           0.98      1177
         macro avg       0.98      0.98      0.98      1177
      weighted avg       0.98      0.98      0.98      1177
```

(2) Logistic Regressin

1) 전체 변수

In [41]:
```
m2_1 = LogisticRegression()

m2_1.fit(x_train, y_train)
p2_1 = m2_1.predict(x_val)
```

```
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 [42]:
```
print('accuracy :',accuracy_score(y_val, p2_1))
print('='*60)
print(confusion_matrix(y_val, p2_1))
print('='*60)
print(classification_report(y_val, p2_1))
```

```
accuracy : 0.9881053525913339
============================================================
[[230   0   0   0   0   0]
 [  0 203   6   0   0   0]
 [  0   6 212   0   0   0]
 [  0   0   0 180   0   0]
 [  0   0   0   0 165   0]
 [  0   0   0   2   0 173]]
============================================================
                    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
          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                           0.99      1177
         macro avg       0.99      0.99      0.99      1177
      weighted avg       0.99      0.99      0.99      1177
```

2) (옵션)적절히 선택한 변수

- 변수 중요도 상위 100로 모델링 해보기

In [43]:
```
feature100 = r.loc[:99, 'feature_name']
```

In [44]:
```
x_train100 = x_train[feature100]
x_val100 = x_val[feature100]
```

In [45]:
```
m2_2 = LogisticRegression()

m2_2.fit(x_train100, y_train)
p2_2 = m2_2.predict(x_val100)
```

```
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 [46]:
```
print('accuracy :',accuracy_score(y_val, p2_2))
print('='*60)
print(confusion_matrix(y_val, p2_2))
print('='*60)
print(classification_report(y_val, p2_2))
```

```
accuracy : 0.9524214103653356
============================================================
[[230   0   0   0   0   0]
 [  0 179  30   0   0   0]
 [  0  19 199   0   0   0]
 [  0   0   0 176   1   3]
 [  0   0   0   0 165   0]
 [  0   0   0   3   0 172]]
============================================================
                    precision    recall  f1-score   support

           LAYING       1.00      1.00      1.00       230
          SITTING       0.90      0.86      0.88       209
         STANDING       0.87      0.91      0.89       218
          WALKING       0.98      0.98      0.98       180
WALKING_DOWNSTAIRS       0.99      1.00      1.00       165
  WALKING_UPSTAIRS       0.98      0.98      0.98       175

          accuracy                           0.95      1177
         macro avg       0.96      0.95      0.96      1177
      weighted avg       0.95      0.95      0.95      1177
```

(3) SVM

1) kernel = 'rbf'

In [47]:
```
#매개변수 C, gamma, kernel 모두 기본값
m3_1 = SVC(C=1, gamma=0.01, kernel = 'rbf', random_state=2022)
m3_1.fit(x_train, y_train)
p3_1 = m3_1.predict(x_val)
```

In [48]:
```
print('accuracy :',accuracy_score(y_val, p3_1))
print('='*60)
print(confusion_matrix(y_val, p3_1))
print('='*60)
print(classification_report(y_val, p3_1))
```

```
accuracy : 0.9762107051826678
============================================================
[[230   0   0   0   0   0]
 [  0 196  13   0   0   0]
 [  0  13 205   0   0   0]
 [  0   0   0 180   0   0]
 [  0   0   0   0 165   0]
 [  0   0   0   2   0 173]]
============================================================
                    precision    recall  f1-score   support

           LAYING       1.00      1.00      1.00       230
          SITTING       0.94      0.94      0.94       209
         STANDING       0.94      0.94      0.94       218
          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                           0.98      1177
        macro avg       0.98      0.98      0.98      1177
     weighted avg       0.98      0.98      0.98      1177
```

2) kernel = 'linear'

In [49]:
```
m3_2 = SVC(C=1, gamma=0.01, kernel = 'linear', random_state=2022)
m3_2.fit(x_train, y_train)
p3_2 = m3_2.predict(x_val)
```

In [50]:
```
print('accuracy :',accuracy_score(y_val, p3_2))
print('='*60)
print(confusion_matrix(y_val, p3_2))
print('='*60)
print(classification_report(y_val, p3_2))
```

```
accuracy : 0.9864061172472387
============================================================
[[230   0   0   0   0   0]
 [  0 200   9   0   0   0]
 [  0   7 211   0   0   0]
 [  0   0   0 180   0   0]
 [  0   0   0   0 165   0]
 [  0   0   0   0   0 175]]
============================================================
                    precision    recall  f1-score   support

           LAYING       1.00      1.00      1.00       230
          SITTING       0.97      0.96      0.96       209
         STANDING       0.96      0.97      0.96       218
          WALKING       1.00      1.00      1.00       180
WALKING_DOWNSTAIRS       1.00      1.00      1.00       165
  WALKING_UPSTAIRS       1.00      1.00      1.00       175

         accuracy                           0.99      1177
        macro avg       0.99      0.99      0.99      1177
     weighted avg       0.99      0.99      0.99      1177
```

3) 하이퍼파라미터 튜닝

- test 전체 수행하기(12 ~ 15분 소요)

In [51]:
```
# 딕셔너리 형태로 관심있는 매개변수의 그리드 설정하기
param_grid = {'C': [0.01, 1, 100],
              'gamma': [0.0001, 0.001, 0.1, 1],
               'kernel': ['linear', 'rbf'] }

#생성
# refit=True 가 default 임. True이면 가장 좋은 파라미터 설정으로 재학습시킴.
model_svc = SVC()
m3_3= GridSearchCV (model_svc, param_grid, return_train_score=True) #cross-validation 옵션 추가

#학습
# train data로 param_grid의 하이퍼 파라미터들을 순차적으로 학습/평가 .
m3_3.fit(x_train, y_train)

#결과
# GridSearchCV 결과 추출하여 DataFrame으로 변환
scores_df = pd.DataFrame(m3_3.cv_results_)
```

In [52]:
```
print('최적파라미터:', m3_3.best_params_)
print('='*60)
print('최고성능:', m3_3.best_score_)
print('='*60)
```

```
최적파라미터: {'C': 1, 'gamma': 0.0001, 'kernel': 'linear'}
============================================================
최고성능: 0.98448187758609
============================================================
```

- 예측 및 평가

In [53]:
```
p3_3 = m3_3.predict(x_val)
```

In [54]:
```
print('accuracy :',accuracy_score(y_val, p3_3))
print('='*60)
print(confusion_matrix(y_val, p3_3))
print('='*60)
print(classification_report(y_val, p3_3))
```

```
accuracy : 0.9864061172472387
============================================================
[[230   0   0   0   0   0]
 [  0 200   9   0   0   0]
 [  0   7 211   0   0   0]
 [  0   0   0 180   0   0]
 [  0   0   0   0 165   0]
 [  0   0   0   0   0 175]]
============================================================
                    precision    recall  f1-score   support

           LAYING       1.00      1.00      1.00       230
          SITTING       0.97      0.96      0.96       209
         STANDING       0.96      0.97      0.96       218
          WALKING       1.00      1.00      1.00       180
WALKING_DOWNSTAIRS       1.00      1.00      1.00       165
  WALKING_UPSTAIRS       1.00      1.00      1.00       175

         accuracy                           0.99      1177
        macro avg       0.99      0.99      0.99      1177
     weighted avg       0.99      0.99      0.99      1177
```

In [ ]: