

Ensemble

PinkLAB Edu

10 November, 2025

Table of Contents

1	양상을 기법	4
1.1	양상을 개요	4
1.2	voting	4
1.3	bagging.....	5
1.4	최종 결정에서 하드보팅.....	5
1.5	최종 결정에서 소프트보팅	6
1.6	랜덤포레스트 Random Forest.....	6
1.7	랜덤포레스트	7
2	HAR, Human Activity Recognition	8
2.1	IMU 센서를 활용해서 사람의 행동을 인식하는 실험	8
2.2	폰에 있는 가속도/자이로 센서 사용	8
2.3	데이터의 공식 경로.....	9
2.4	PinkWink's Github에 옮겨둠	9
2.5	데이터 소개	10
2.6	데이터의 특성.....	10
2.7	데이터의 클래스	11
2.8	시간영역의 데이터를 직접 사용하는 것은 어렵다	14
2.9	머신러닝을 이용한 행동 인식 연구는 꽤 발전했다	14
2.10	데이터 읽기	15
2.11	특성만 무려 561개의 위력	15
2.12	대충 이런 내용들	16
2.13	일단 X 데이터만	16
2.14	수업 2주 만에 우리는 대용량(^^) 데이터를 다루게 되었음	17
2.15	y 데이터 읽어 오기	17
2.16	각 액션별 데이터의 수	18
2.17	각 라벨별 정의.....	18
2.18	뭐 간단하게 결정나무?	18
2.19	max_depth를 다양하게 하기 위해 GridSearchCV 이용	19
2.20	max_depth 8이 좋다고 함	19

2.21	max_depth별로 표로 성능을 정리.....	20
2.22	실제 test 데이터에서의 결과	20
2.23	아무튼, 우리의 베스트 모델의 결과는.....	20
2.24	랜덤포레스트 적용.....	21
2.25	np.ravel()	21
2.25.1	샘플 데이터	22
2.25.2	reshape 적용.....	22
2.25.3	이번엔 다차원 샘플데이터	22
2.25.4	reshape 는?.....	22
2.25.5	ravel 은?	22
2.25.6	결과 정리를 위한 작업	23
2.25.7	성능이 좋음	23
2.25.8	best 모델	23
2.25.9	test 데이터에 적용	24
2.25.10	중요 특성 확인.....	24
2.25.11	주요 특성 관찰.....	25
2.25.12	주요 20개 특성.....	25
2.25.13	20개 특성만 가지고 다시 성능 확인	26

1 양상블 기법

1.1 양상블 개요



양상블학습을통한 분류

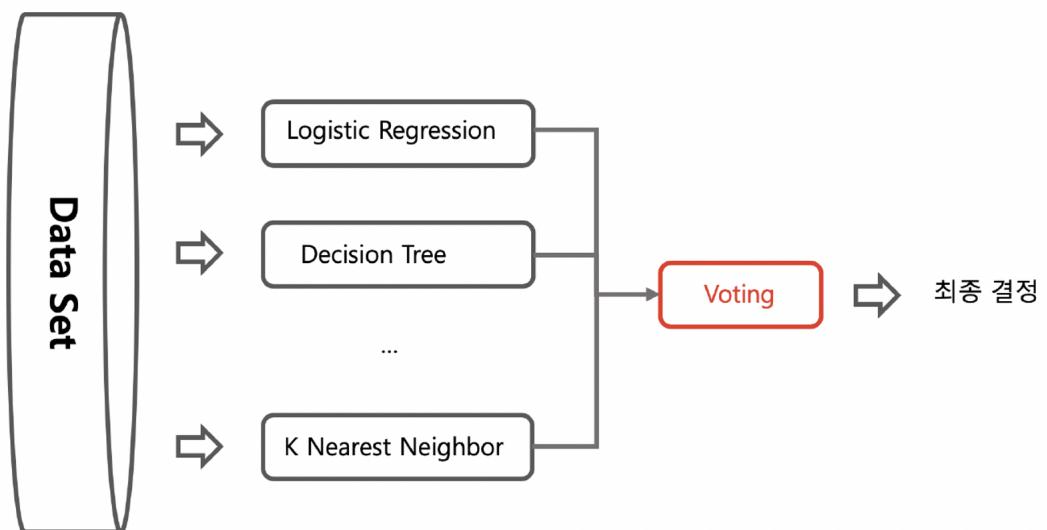
- 여러 개의 분류기를 생성하고 그 예측을 결합하여 정확한 최종 예측을 기대하는 기법

양상블학습의 목표

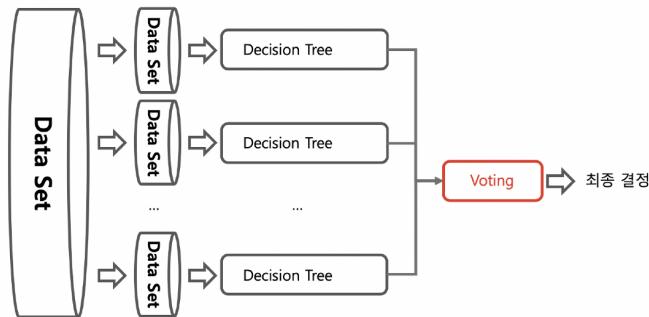
- 다양한 분류기의 예측 결과를 결합함으로써 단일 분류기보다 신뢰성이 높은 예측 값을 얻는 것

● ● 현재 정형데이터를 대상으로 하는 분류기에서는 양상블 기법이 뛰어난 성과를 보여주고 있음 ● ●

1.2 voting

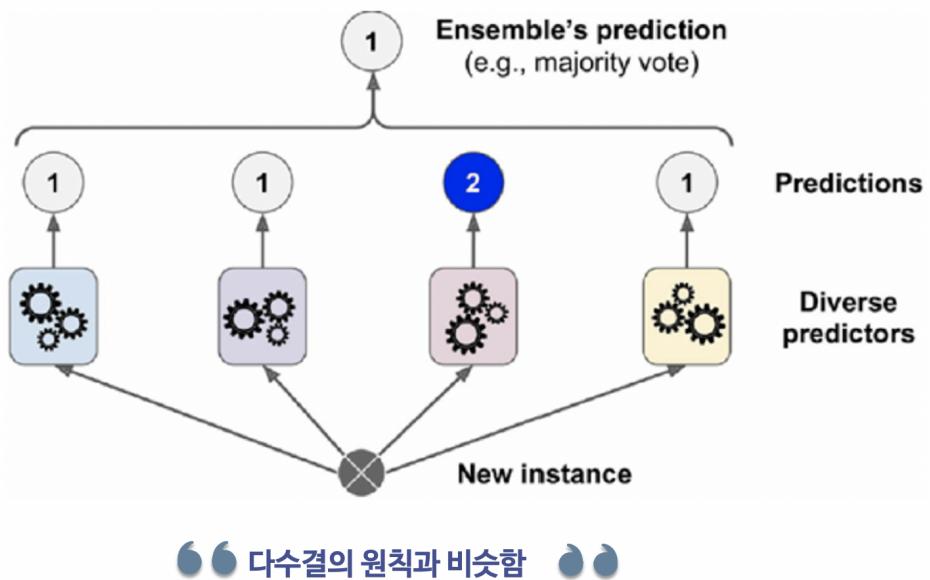


1.3 bagging



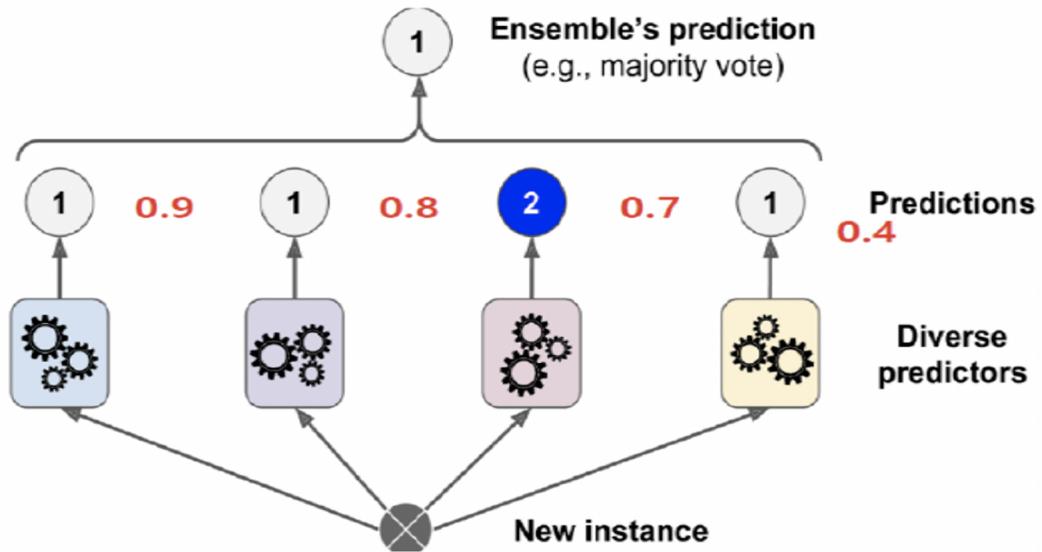
- bagging의 경우 데이터를 중복을 하여 샘플링하고 그 각각의 데이터에 같은 알고리즘을 적용해서 결과를 투표로 결정함
- 각각의 분류기에 데이터를 각각 샘플링해서 추출하는 방식을 **부트스트래핑(bootstrapping)** 분할 방식이라고 함

1.4 최종 결정에서 하드보팅

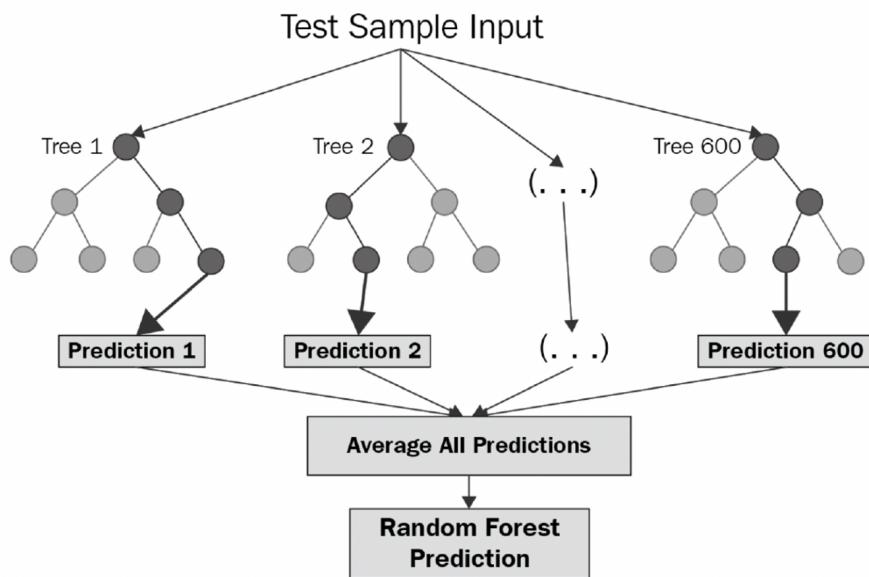


● ● 다수결의 원칙과 비슷함 ● ●

1.5 최종 결정에서 소프트보팅



1.6 랜덤포레스트 Random Forest



1.7 랜덤포레스트



같은 알고리즘으로 구현하는 배깅(Bagging)의 대표적인 방법



양상블 방법 중에서 비교적 속도가 빠르며 다양한 영역에서 높은 성능을 보여주고 있음



부트스트래핑은 여러 개의 작은 데이터 셋을 중첩을 허용해서 만드는 것



랜덤 포레스트는 결정 나무를 기본으로 함

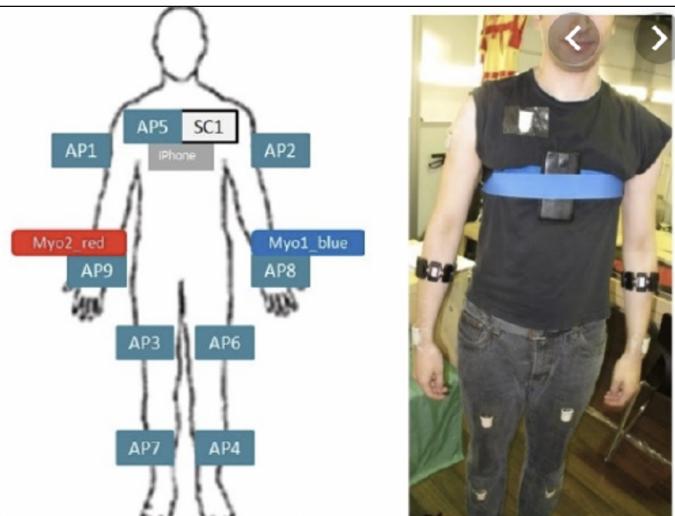


부트스크래핑으로 샘플링된 데이터마다 결정나무가 예측한 결과를 소프트보팅으로
최종 예측 결론을 얻음

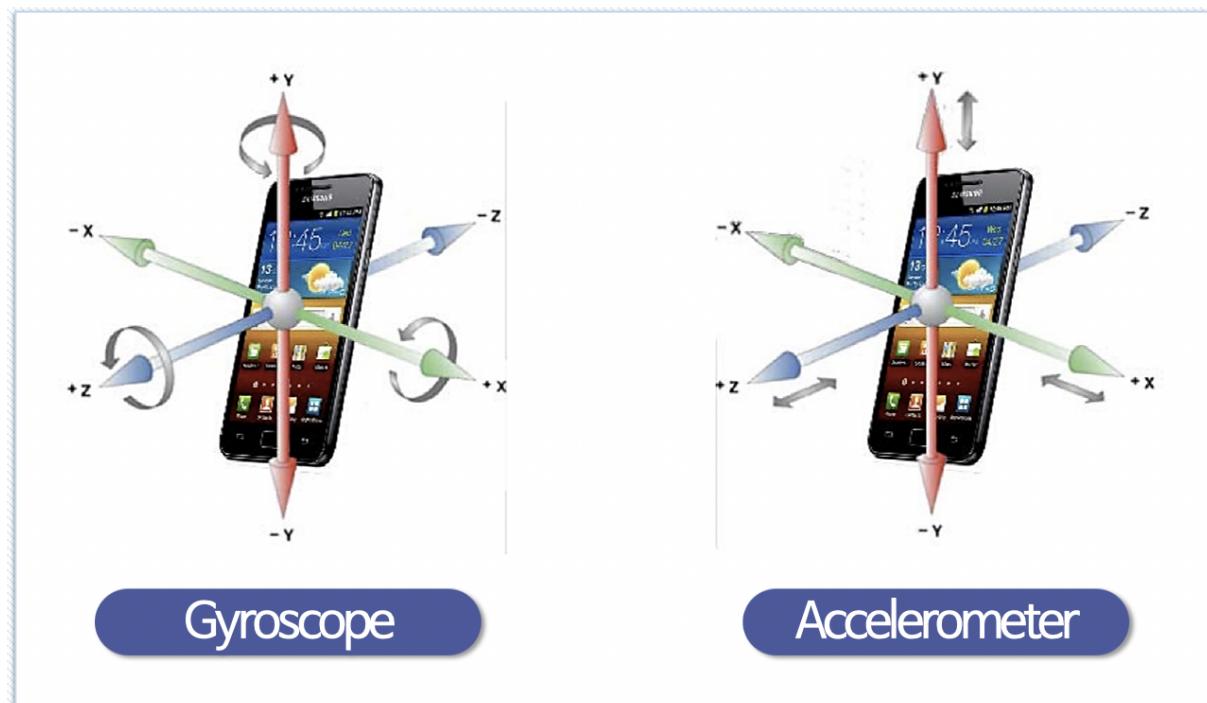
2 HAR, Human Activity Recognition

2.1 IMU 센서를 활용해서 사람의 행동을 인식하는 실험

mount	Required data-measurement-packages per run
9	ActivePAL™ (AP1-9)
1	SmartCardia (SC1)
2	Myo-Armband (Myo1_blue, Myo2_red)
1	iPhone®
1	BPMs™(pressure mattress)
1	memorycard for handcamera
4	memorycards for wall cameras



2.2 폰에 있는 가속도/자이로 센서 사용



2.3 데이터의 공식 경로



Human Activity Recognition Using Smartphones Data Set

[Download: Data Folder](#) [Data Set Description](#)

Abstract: Human Activity Recognition database built from the recordings of 30 subjects performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors.

Data Set Characteristics:	Multivariate, Time-Series	Number of Instances:	10299	Area:	Computer
Attribute Characteristics:	N/A	Number of Attributes:	561	Date Donated:	2012-12-10
Associated Tasks:	Classification, Clustering	Missing Values?	N/A	Number of Web Hits:	896522

Source:

Jorge L. Reyes-Ortiz(1,2), Davide Anguita(1), Alessandro Ghio(1), Luca Oneto(1) and Xavier Parra(2)
 1 - SmartLab – University of Genoa Systems Laboratory
 DITDI – Università degli Studi di Genova, Genoa (I-16145), Italy.
 2 - CEPtD - Technical Research Centre for Dependency Care and Autonomous Living
 Universitat Politècnica de Catalunya (BarcelonaTech), Vilanova i la Geltrú (08800), Spain
 activityrecognition @ smartlab.ws

- <https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones>

2.4 PinkWink's Github에 옮겨둠

The screenshot shows a GitHub repository page for "ML_tutorial / dataset / HAR_dataset /". The repository has 1 star, 2 forks, and 1 issue. The main page displays a list of files added by "PinkWink" (represented by a bear icon) just now. The files listed are: test, train, .DS_Store, README.txt, UCI HAR Dataset.names, activity_labels.txt, features.txt, and features_info.txt. All files were added 1 minute ago.

File	Description	Time Ago
test	HAR data add	1 minute ago
train	HAR data add	1 minute ago
.DS_Store	HAR data add	1 minute ago
README.txt	HAR data add	1 minute ago
UCI HAR Dataset.names	HAR data add	1 minute ago
activity_labels.txt	HAR data add	1 minute ago
features.txt	HAR data add	1 minute ago
features_info.txt	HAR data add	1 minute ago

2.5 데이터 소개

1 UCI HAR 데이터셋은 스마트폰을 장착한 사람의 행동을 관찰한 데이터임

→ 실험 대상 : 19~48세 연령의 30명의 자원 봉사자를 모집하여 수행

2 허리에 스마트 폰 (Samsung Galaxy S II)을 착용하여 50Hz의 주파수로 데이터를 얻음

→ 6 가지 활동 (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING)을 수행함

→ 내장 된 가속도계와 자이로 스코프를 사용하여 50Hz의 일정한 속도로 3 축 선형 가속 및 3 축 각속도를 캡처

3 실험은 데이터를 수동으로 라벨링하기 위해 비디오로 기록함

→ 획득 한 데이터 세트는 무작위로 두 세트로 분할되었음

→ 훈련 데이터 생성을 위해 자원 봉사자의 70 %가 선택되었고 테스트 데이터는 30%가 선정

4 중력 및 신체 운동 성분을 갖는 센서 가속 신호는 버터 워스 저역 통과 필터를 사용하여 신체 가속 및 중력으로 분리함

→ 중력은 저주파 성분만을 갖는 것으로 가정하고 0.3Hz 차단 주파수를 가진 필터가 사용

2.6 데이터의 특성

✓ 가속도계로부터의 3 축 가속도 (총 가속도) 및 추정 된 신체 가속도

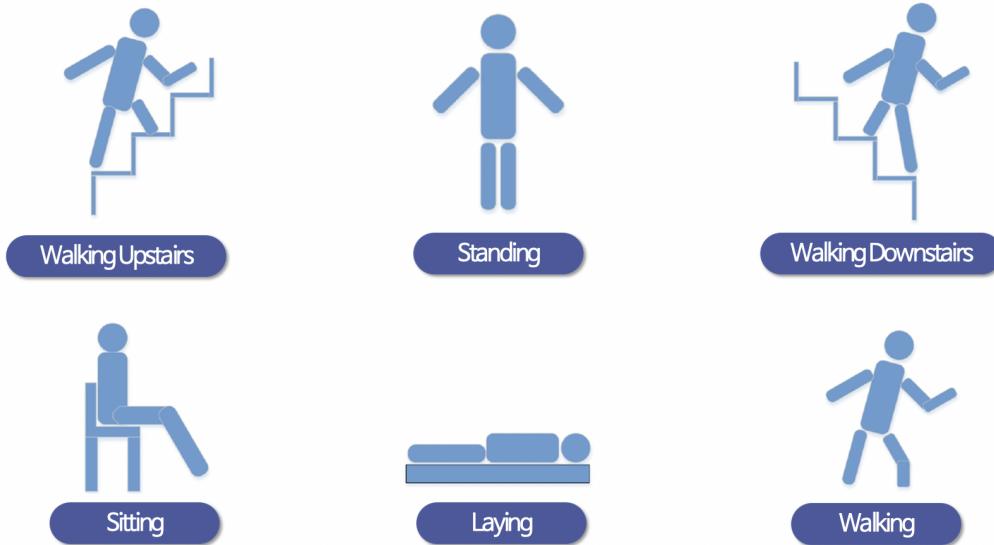
✓ 자이로 스코프의 3 축 각속도

✓ 시간 및 주파수 영역 변수가 포함 된 561 기능 벡터

✓ 활동 라벨

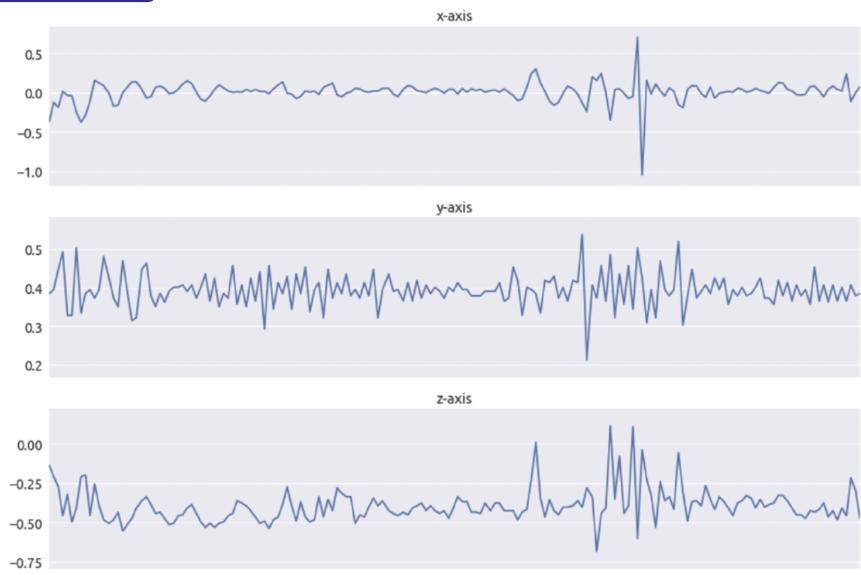
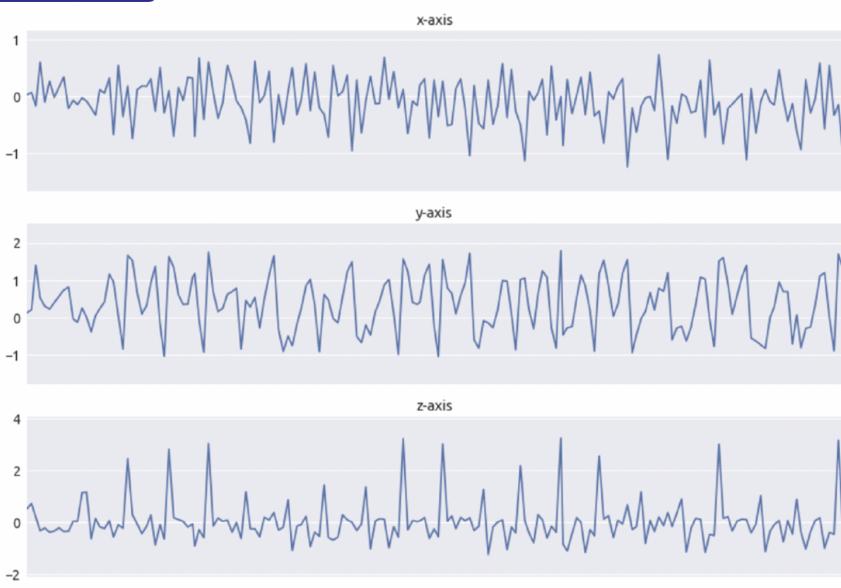
✓ 실험을 수행 한 대상의 식별자

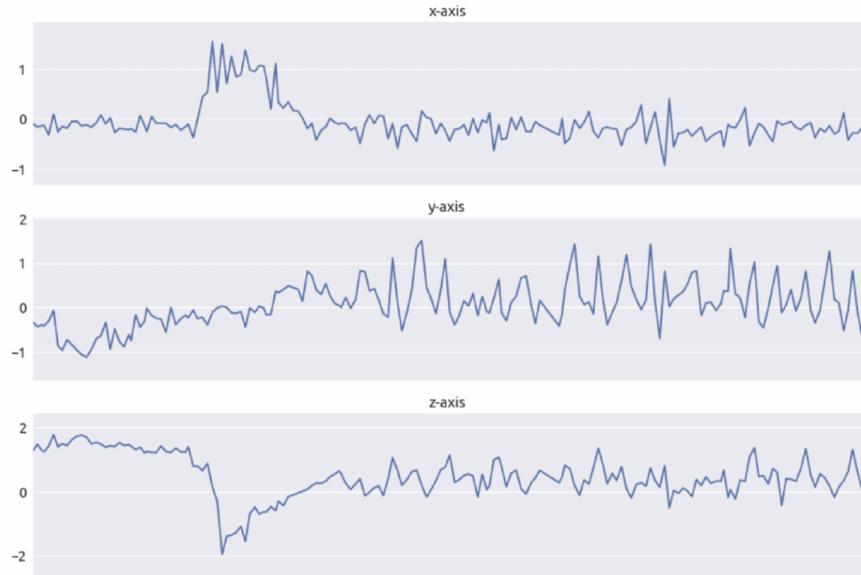
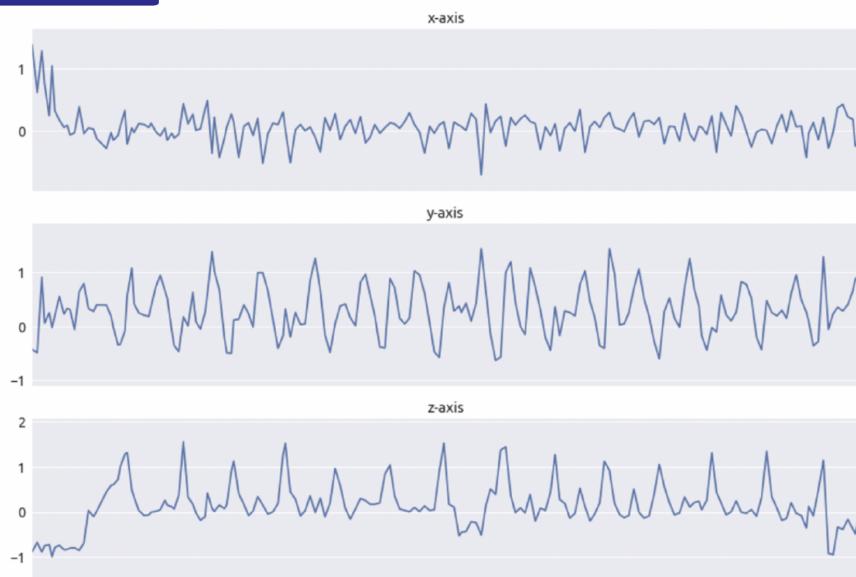
2.7 데이터의 클래스



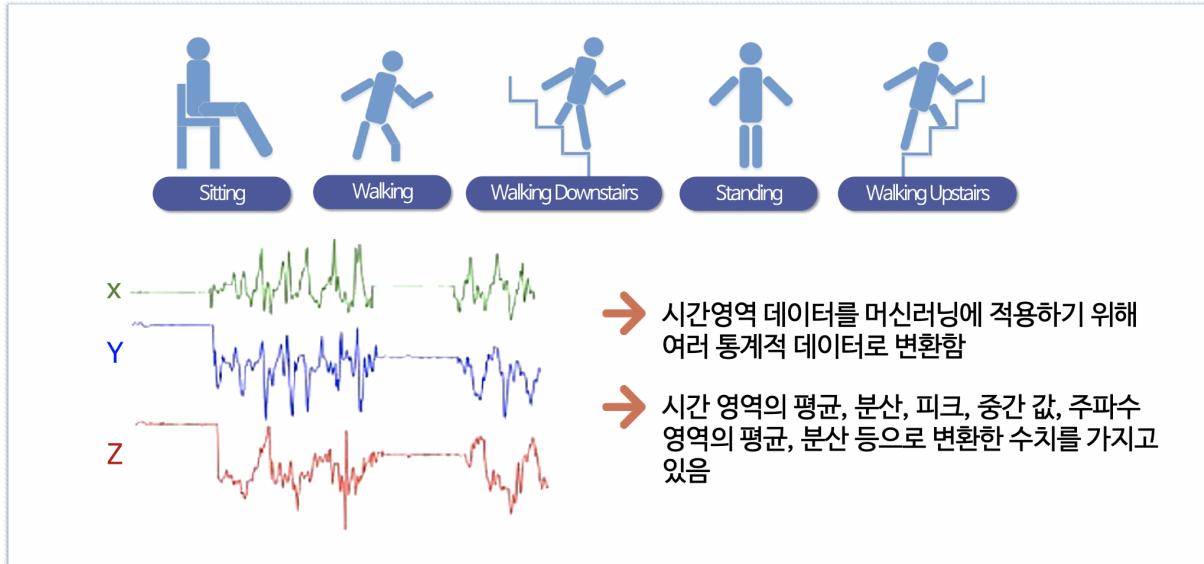
- https://www.youtube.com/watch?v=XOEN9W05_4A



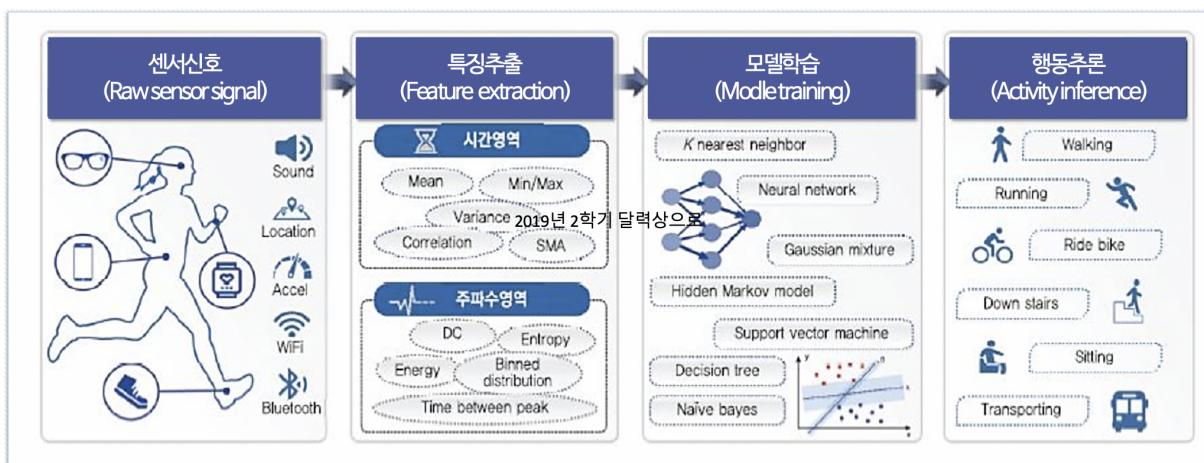
STANDING**WALKING**

WALKING DOWNSTAIRS**WALKING UPSTAIRS**

2.8 시간영역의 데이터를 직접 사용하는 것은 어렵다



2.9 머신러닝을 이용한 행동 인식 연구는 꽤 발전했다



2.10 데이터 읽기

```
import pandas as pd
import matplotlib.pyplot as plt
✓ 0.5s
```

Python

```
url = (
    "https://github.com/PinkWink/ML_tutorial/raw/refs/heads/master/"
    + "dataset/HAR_dataset/features.txt"
)

feature_name_df = pd.read_csv(
    url, header=None, sep="\s+", names=["column_index", "column_name"]
)
feature_name_df.head()
```

✓ 0.8s

Python

<:7: SyntaxWarning: invalid escape sequence '\s'
<:7: SyntaxWarning: invalid escape sequence '\s'
`/tmp/ipykernel_40470/2322774443.py:7: SyntaxWarning: invalid escape sequence '\s'`
`url, header=None, sep="\s+", names=["column_index", "column_name"]`

column_index	column_name
0	tBodyAcc-mean()-X
1	tBodyAcc-mean()-Y
2	tBodyAcc-mean()-Z
3	tBodyAcc-std()-X
4	tBodyAcc-std()-Y

- 조심! 지금은 특성 이름만 읽었을 뿐

2.11 특성만 무려 561개의 위력

```
len(feature_name_df)
✓ 0.0s
```

Python

561

2.12 대충 이런 내용들

```
feature_name = feature_name_df.iloc[:, 1].tolist()
feature_name[:10]

✓ 0.0s
```

Python

['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']

2.13 일단 X 데이터만

```
X_train_url = (
    "https://github.com/PinkWink/ML_tutorial/raw/refs/heads/master/"
    + "dataset/HAR_dataset/train/X_train.txt"
)
X_test_url = (
    "https://github.com/PinkWink/ML_tutorial/raw/refs/heads/master/"
    + "dataset/HAR_dataset/test/X_test.txt"
)

X_train = pd.read_csv(X_train_url, sep="\s+", header=None)
X_test = pd.read_csv(X_test_url, sep="\s+", header=None)
```

✓ 12.4s

Python

- X_train.txt의 용량이 63MB.

2.14 수업 2주 만에 우리는 대용량(^^) 데이터를 다루게 되었음

```
X_train.columns = feature_name
X_test.columns = feature_name
X_train.head()
```

✓ 0.0s

Python

	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()
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990

5 rows × 561 columns

2.15 y 데이터 읽어 오기

```
y_train_url = (
    "https://github.com/PinkWink/ML_tutorial/raw/refs/heads/master/"
    + "dataset/HAR_dataset/train/y_train.txt"
)
y_test_url = (
    "https://github.com/PinkWink/ML_tutorial/raw/refs/heads/master/"
    + "dataset/HAR_dataset/test/y_test.txt"
)

y_train = pd.read_csv(y_train_url, sep="\s+", header=None, names=["action"])
y_test = pd.read_csv(y_test_url, sep="\s+", header=None, names=["action"])
```

✓ 1.3s

Python

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

✓ 0.0s

Python

((7352, 561), (2947, 561), (7352, 1), (2947, 1))

2.16 각 액션별 데이터의 수

```
y_train["action"].value_counts()
```

✓ 0.0s

Python

```
action
6    1407
5    1374
4    1286
1    1226
2    1073
3     986
Name: count, dtype: int64
```

2.17 각 라벨별 정의



1. Walking



2. Walking Upstairs



3. Walking Downstairs



4. Sitting



5. Standing



6. Laying

2.18 뭐 간단하게 결정나무?

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

dt_clf = DecisionTreeClassifier(random_state=13, max_depth=4)
dt_clf.fit(X_train, y_train)
pred = dt_clf.predict(X_test)

accuracy_score(y_test, pred)
```

✓ 1.9s

Python

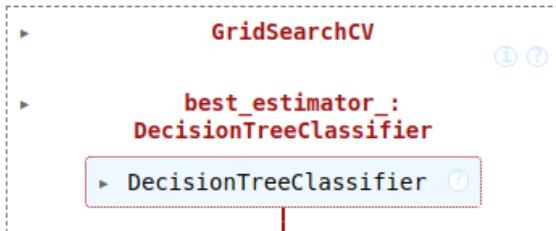
0.8096369189005769

2.19 max_depth를 다양하게 하기 위해 GridSearchCV 이용

```
from sklearn.model_selection import GridSearchCV  
  
params = {"max_depth": [6, 8, 10, 12, 16, 20, 24]}  
  
grid_cv = GridSearchCV(  
    dt_clf, param_grid=params, scoring="accuracy", cv=5, return_train_score=True  
)  
  
grid_cv.fit(X_train, y_train)
```

✓ 1m 38.7s

Python



2.20 max_depth 80| 좋다고 함

```
grid_cv.best_score_  
✓ 0.0s  
  
np.float64(0.8541974777674495)
```

Python

```
grid_cv.best_params_  
✓ 0.0s  
{'max_depth': 8}
```

Python

2.21 max_depth별로 표로 성능을 정리

```
cv_results_df = pd.DataFrame(grid_cv.cv_results_)
cv_results_df[["param_max_depth", "mean_test_score", "mean_train_score"]]
```

✓ 0.0s

Python

	param_max_depth	mean_test_score	mean_train_score
0	6	0.843444	0.944879
1	8	0.854197	0.982692
2	10	0.847125	0.993369
3	12	0.842503	0.997212
4	16	0.839510	0.999660
5	20	0.840597	0.999966
6	24	0.839781	1.000000

- train과 test의 score 차이가 있음. 과적합일까?

2.22 실제 test 데이터에서의 결과

```
max_depths = [6, 8, 10, 12, 16, 20, 24]

for depth in max_depths:
    dt_clf = DecisionTreeClassifier(max_depth=depth, random_state=13)
    dt_clf.fit(X_train, y_train)
    pred = dt_clf.predict(X_test)
    accuracy = accuracy_score(y_test, pred)
    print("Max_Depth = ", depth, ", Accuracy = ", accuracy)
```

✓ 25.4s

Python

```
Max_Depth = 6 , Accuracy = 0.8554462164913471
Max_Depth = 8 , Accuracy = 0.8734306073973532
Max_Depth = 10 , Accuracy = 0.8615541228367831
Max_Depth = 12 , Accuracy = 0.8680013573125213
Max_Depth = 16 , Accuracy = 0.8632507634882932
Max_Depth = 20 , Accuracy = 0.8527315914489311
Max_Depth = 24 , Accuracy = 0.8527315914489311
```

2.23 아무튼, 우리의 베스트 모델의 결과는

```
best_df_clf = grid_cv.best_estimator_
pred1 = best_df_clf.predict(X_test)

accuracy_score(y_test, pred1)
```

✓ 0.0s

Python

```
0.8734306073973532
```

2.24 랜덤포레스트 적용

```

import numpy as np

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

params = {
    "max_depth": [6, 8, 10],
    "n_estimators": [50, 100, 200],
    "min_samples_leaf": [8, 12],
    "min_samples_split": [8, 12],
}

rf_clf = RandomForestClassifier(random_state=13, n_jobs=-1)
grid_cv = GridSearchCV(rf_clf, param_grid=params, cv=2, n_jobs=-1)
grid_cv.fit(X_train, np.ravel(y_train))

```

✓ 1m 24.75 Python

- 여기서 n_jobs 옵션을 높여주면 CPU의 코어를 보다 병렬로 활용함. Core가 많으면 n_jobs를 높이면 속도가 빨라짐

2.25 np.ravel()



[Home](#) > NumPy reference > ... > Array manipulation routines > [numpy.ravel](#)

numpy.ravel

`numpy.ravel(a, order='C')`

[\[source\]](#)

Return a contiguous flattened array.

A 1-D array, containing the elements of the input, is returned. A copy is made only if needed.

As of NumPy 1.10, the returned array will have the same type as the input array. (for example, a masked array will be returned for a masked array input)

- 다 차원의 데이터를 1차원의 데이터로 변형시켜줌

2.25.1 샘플 데이터

```
import numpy as np

x = np.array([[1, 2, 3], [4, 5, 6]])

np.ravel(x)
✓ 0.0s
```

Python

array([1, 2, 3, 4, 5, 6])

2.25.2 reshape 적용

```
x.reshape(-1)
✓ 0.0s
```

Python

array([1, 2, 3, 4, 5, 6])

2.25.3 이번엔 다차원 샘플데이터

```
x = np.random.randint(1, 10, (3, 3))
x
✓ 0.0s
```

Python

array([[5, 9, 6],
 [6, 5, 6],
 [3, 5, 9]])

2.25.4 reshape 는?

```
x.reshape(-1)
✓ 0.0s
```

Python

array([5, 9, 6, 6, 5, 6, 3, 5, 9])

2.25.5 ravel 은?

```
np.ravel(x)
✓ 0.0s
```

Python

array([5, 9, 6, 6, 5, 6, 3, 5, 9])

2.25.6 결과 정리를 위한 작업

```
cv_results_df = pd.DataFrame(grid_cv.cv_results_)
cv_results_df.columns
Python
```

Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
 'param_max_depth', 'param_min_samples_leaf', 'param_min_samples_split',
 'param_n_estimators', 'params', 'split0_test_score',
 'split1_test_score', 'mean_test_score', 'std_test_score',
 'rank_test_score'],
 dtype='object')

2.25.7 성능이 좋음

```
target_col = [
    "rank_test_score",
    "mean_test_score",
    "param_n_estimators",
    "param_max_depth",
]
cv_results_df[target_col].sort_values("rank_test_score").head()
Python
```

	rank_test_score	mean_test_score	param_n_estimators	param_max_depth
35	1	0.912541	200	10
32	1	0.912541	200	10
13	3	0.912405	100	8
28	3	0.912405	100	10
25	3	0.912405	100	10

2.25.8 best 모델

```
grid_cv.best_params_
Python
```

{'max_depth': 10,
 'min_samples_leaf': 12,
 'min_samples_split': 8,
 'n_estimators': 200}

```
grid_cv.best_score_
Python
```

np.float64(0.9125408052230686)

2.25.9 test 데이터에 적용

```

rf_clf_best = grid_cv.best_estimator_
rf_clf_best.fit(X_train, np.ravel(y_train))

pred1 = rf_clf_best.predict(X_test)

accuracy_score(y_test, pred1)
✓ 4.7s
0.9175432643366135
    
```

Python

2.25.10 중요 특성 확인

```

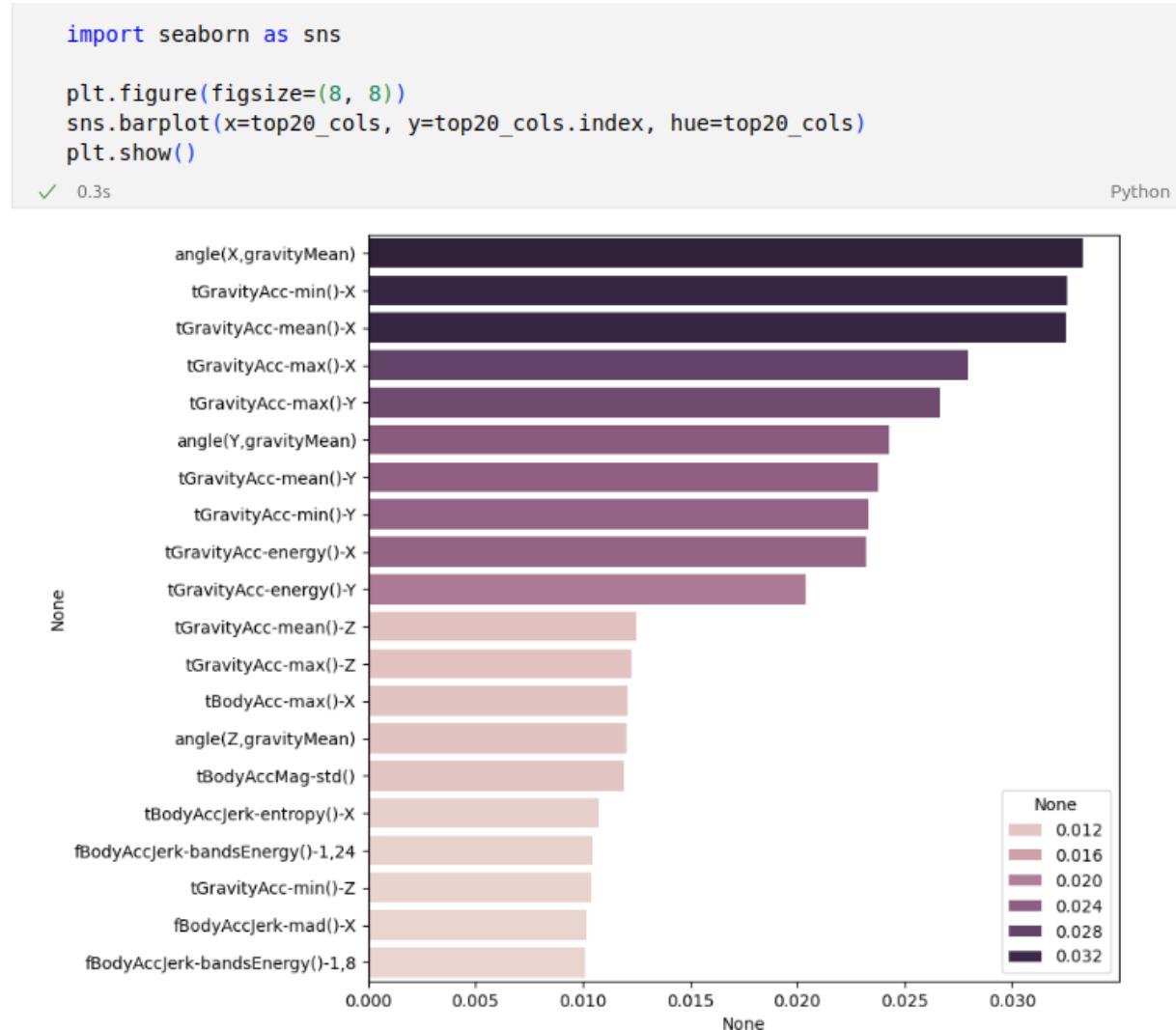
best_cols_values = rf_clf_best.feature_importances_
best_cols = pd.Series(best_cols_values, index=X_train.columns)
top20_cols = best_cols.sort_values(ascending=False)[:20]
top20_cols
✓ 0.0s
    
```

특성	중요도
angle(X,gravityMean)	0.033315
tGravityAcc-min()-X	0.032563
tGravityAcc-mean()-X	0.032541
tGravityAcc-max()-X	0.027953
tGravityAcc-max()-Y	0.026627
angle(Y,gravityMean)	0.024259
tGravityAcc-mean()-Y	0.023765
tGravityAcc-min()-Y	0.023294
tGravityAcc-energy()-X	0.023224
tGravityAcc-energy()-Y	0.020372
tGravityAcc-mean()-Z	0.012478
tGravityAcc-max()-Z	0.012247
tBodyAcc-max()-X	0.012110
angle(Z,gravityMean)	0.012056
tBodyAccMag-std()	0.011938
tBodyAccJerk-entropy()-X	0.010735
fBodyAccJerk-bandsEnergy()-1,24	0.010430
tGravityAcc-min()-Z	0.010385
fBodyAccJerk-mad()-X	0.010179
fBodyAccJerk-bandsEnergy()-1,8	0.010123

dtype: float64

- 각 특성들의 중요도가 개별적으로 높지 않다

2.25.11 주요 특성 관찰



2.25.12 주요 20개 특성

```
top20_cols.index
✓ 0.0s
```

```
Index(['angle(X,gravityMean)', 'tGravityAcc-min()-X', 'tGravityAcc-mean()-X',
       'tGravityAcc-max()-X', 'tGravityAcc-max()-Y', 'angle(Y,gravityMean)',
       'tGravityAcc-mean()-Y', 'tGravityAcc-min()-Y', 'tGravityAcc-energy()-X',
       'tGravityAcc-energy()-Y', 'tGravityAcc-mean()-Z', 'tGravityAcc-max()-Z',
       'tBodyAcc-max()-X', 'angle(Z,gravityMean)', 'tBodyAccMag-std()', 
       'tBodyAccJerk-entropy()-X', 'fBodyAccJerk-bandsEnergy()-1,24',
       'tGravityAcc-min()-Z', 'fBodyAccJerk-mad()-X',
       'fBodyAccJerk-bandsEnergy()-1,8'],
      dtype='object')
```

2.25.13 20개 특성만 가지고 다시 성능 확인

```
X_train_re = X_train[top20_cols.index]  
X_test_re = X_test[top20_cols.index]
```

✓ 0.0s

Python

```
rf_clf_best_re = grid_cv.best_estimator_  
rf_clf_best_re.fit(X_train_re, np.ravel(y_train))
```

```
pred1_re = rf_clf_best_re.predict(X_test_re)
```

```
accuracy_score(y_test, pred1_re)
```

✓ 1.2s

Python

0.8055649813369529

- 561개의 특성보다 20개의 특성만 보면 연산속도가 정말 빠를 것이다. 비록 acc는 포기하더라도~