스마트폰 센서 데이터 기반 모션 분류

김영우

1. 프로젝트 개요 – 요약

[주제] 스마트폰 센서 기반 데이터를 활용한 행동 인식

[주요 기술] 데이터 처리, 머신러닝, 딥러닝

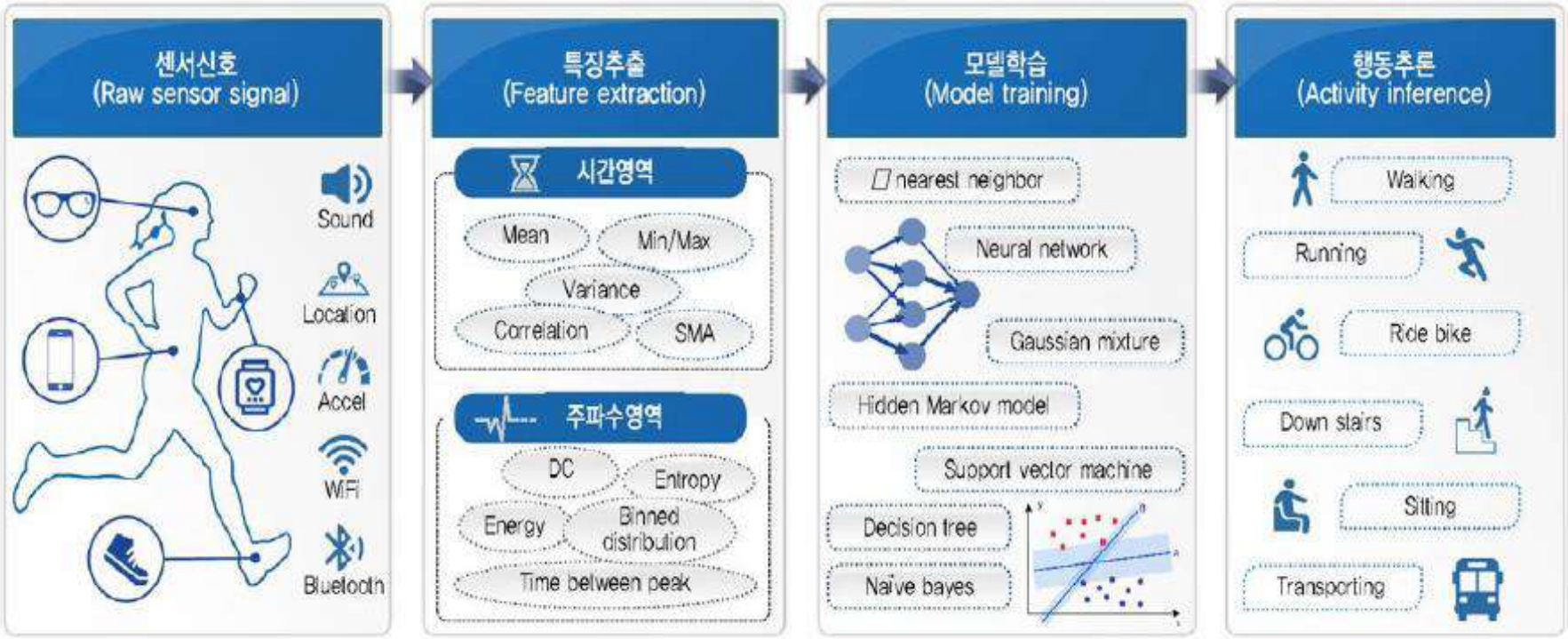
[데이터 출처] UCI Machine Learning Repository

[데이터 구분] Tabular

[중심 사항] 561개 feature에 대한 데이터 탐색
6개 class 관계를 고려한 모델링

2. 데이터 소개 – Domain Knowledge

- 인간 행동 인식 – 다양한 센서를 활용하여 사람의 모션에 관련된 정보를 수집하고 해석하여 행동을 인식하는 기술

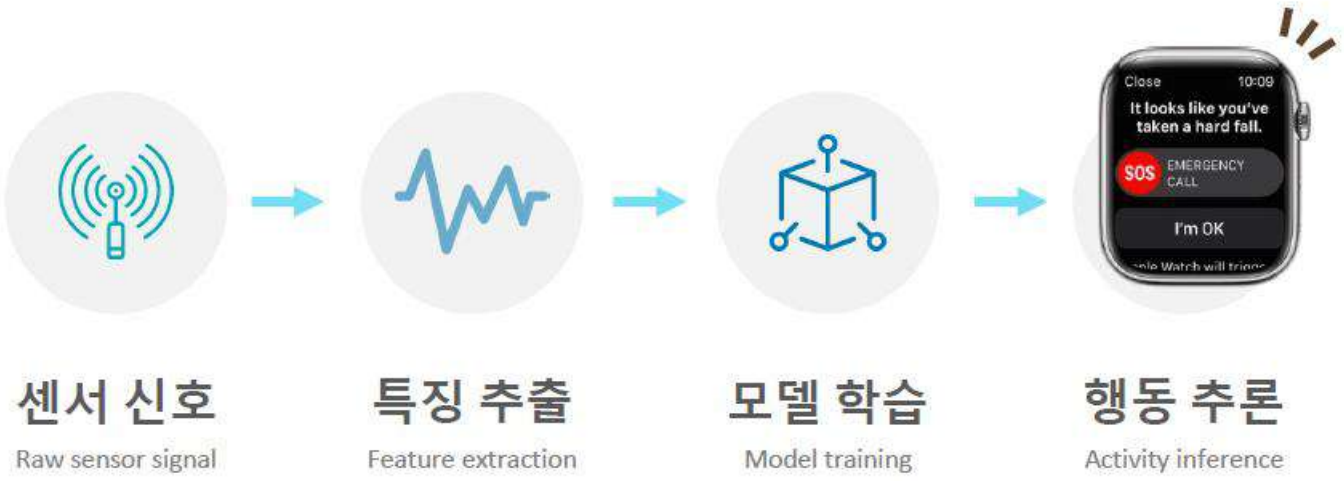


2. 데이터 소개 – Domain Knowledge

행동 추론(Activity Inference)

2. 데이터 소개 – Domain Knowledge

센서 신호 -> 특징 추출까지 해서 보기좋게 Tabular 데이터가 제공됐으므로
우리는 이 데이터를 가지고, 모델을 학습하여 행동을 추론할 수 있는 파이프라인을 만든다.



2. 데이터 소개 - 출처

UCI – Human Activity Recognition

UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Check out the beta version of the new UCI Machine Learning Repository we are currently testing! Contact us if you have any issues, questions, or concerns. Click here to try out the new site.

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:**	1301361

Source:

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

2. 데이터 소개 - 출처


30 volunteers
Wearing Samsung Galaxy S2
Performing 6 posture activities

2. 데이터 소개 – 데이터 형태

data.csv

(row **5881**, column **563**)

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	...	Subject	Activity
1	0.288585	-0.02029	-0.13291	-0.99528		1	
2	0.278419	-0.01641	-0.12352	-0.99825		1	
3	0.279653	-0.01947	-0.11346	-0.99538		1	
4	0.279174	-0.0262	-0.12328	-0.99609		1	
5	0.276629	-0.01657	-0.11536	-0.99814		1	
6	0.277199	-0.0101	-0.10514	-0.99733		1	
7	0.279454	-0.01964	-0.11002	-0.99692		1	
...							

X_feature
562

tBodyAcc-mean()-X
tBodyAcc-mean()-Y
tBodyAcc-mean()-Z
tBodyAcc-std()-X
tBodyAcc-std()-Y
...

Y_label 1

Activity

3. 프로젝트 진행 방식 – 문제 정의

- 업무 : 스마트폰 기반의 센서 데이터를 활용해
  동작을 분류하는 모델을 완성한다.

4. 프로젝트 진행방식 – 과제 핵심 사항

- 너무 많은 Feature들(561개)
  EDA 수행의 어려움 – 모든 feature들에 대해서 다 그래프를 그려야할까?
  모델의 복잡도 증가 – 모든 feature가 모델링에 필요할까?

  Sol. **선택과 집중** – 트리 기반 모델로부터 변수 중요도 추출
  - 상위 N개 변수에 대해서 탐색 및 모델링

- 다중 분류
  6개 class 상호 관련이 있는 Class들
  정적 : Laying, Sitting, Standing
  동적 : Walking, Walking-Up, Walking-Down

  Sol. **단계별 모델링** – 단계 1. 0(정적)과 1(동적)에 대한 분류
  - 단계 2. 정적에 대한 세세한 분류(Laying, Sitting, Standing)와
    동적에 대한 세세한 분류(Walking, Walking-Up, Walking-Down)

5. 프로젝트 전략

- **561개의 변수 중 n개 변수만 추출한다.**

  -> 6개 행동 구분, 정적(0) * 동적(1) 행동 구분, Standing 구분, Laying 구분, Walking 구분,
  Walking_upstairs 구분, Walking_downstairs 구분을 포함해서 모든 관점에서 변수 중요도를
  5개씩 추출하여 하나로 합친 후 저장한다.

- **6개를 다중 분류하는 여러 모델을 만들어본다. (기본 모델링)**

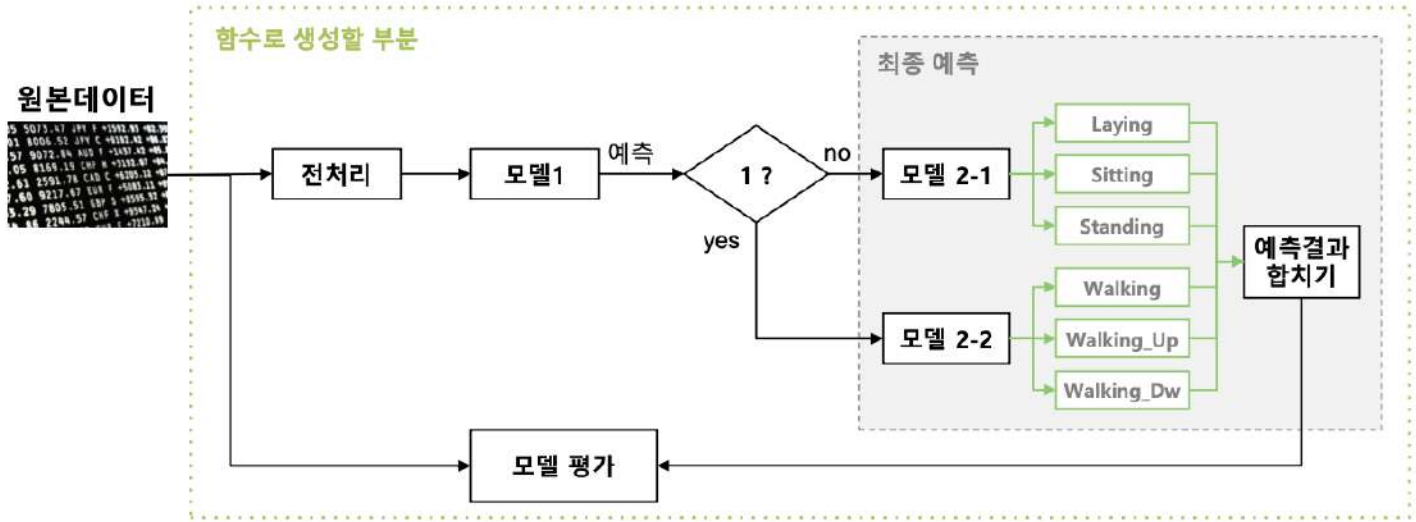
  -> 변수 5개부터 n개까지 하나씩 수행해서 모델의 정확도가 얼마나 나오는지 파악한다.

  -> 변수 몇개를 가지고 어떤 모델을 수행했을 때 가장 정확도가 높게 나오는지 확인한다.

- **정적(0)과 동적(1)를 분류하는 모델 1을 만든다음,**
  정적에 대한 세세한 분류(모델 2-1)와 동적에 대한 세세한 분류(모델 2-2)하는 모델을 만들어본다. (단계적 모델링)

5. 프로젝트 전략

단계적 모델링은 아래 사진과 같이 진행이 될 것이다.



6. 탐색적 데이터 분석

data에 따른 변수들과 shape는 어떤지 알아본다.

```
[6]  1  data.shape

     (5881, 563)
```

```
1  data.head()
```

코드 셀 추가
⌘/Ctrl+M B

	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	tBodyAcc-max()-Y
0	0.288508	-0.009196	-0.103362	-0.988986	-0.962797	-0.967422	-0.989000	-0.962596	-0.965650	-0.929747	-0.554597
1	0.265757	-0.016576	-0.098163	-0.989551	-0.994636	-0.987435	-0.990189	-0.993870	-0.987558	-0.937337	-0.573949
2	0.278709	-0.014511	-0.108717	-0.997720	-0.981088	-0.994008	-0.997934	-0.982187	-0.995017	-0.942584	-0.566451
3	0.289795	-0.035536	-0.150354	-0.231727	-0.006412	-0.338117	-0.273557	0.014245	-0.347916	0.008288	-0.136535
4	0.394807	0.034098	0.091229	0.088489	-0.106636	-0.388502	-0.010469	-0.109680	-0.346372	0.584131	-0.111170

6. 탐색적 데이터 분석

Activity(Target) 클래스 분포가 그래도 비교적 균형하다는 것을 알 수 있다.
정적(LAYING, STANDING, SITTING)과 동적(WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS)으로 구분하면 더욱 균형하다는 것을 알 수 있다.

```
[ ]  1  data['Activity'].unique()

array(['STANDING', 'LAYING', 'WALKING', 'WALKING_DOWNSTAIRS',
       'WALKING_UPSTAIRS', 'SITTING'], dtype=object)
```

```
1  data['Activity'].value_counts()

LAYING                1115
STANDING              1087
SITTING               1032
WALKING                998
WALKING_UPSTAIRS       858
WALKING_DOWNSTAIRS     791
Name: Activity, dtype: int64
```

6. 탐색적 데이터 분석

- 6개 다중 분류했을 때 상위 5개 변수를 확인한다. (관점 1)

```
rom sklearn.model_selection import train_test_split

 = data.drop(['Activity'], axis=1)
 = data['Activity']

rain_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, random_state=42) # stratify는 쓰지 않아도 될 것
```

```
[ ]  train_X.shape, test_X.shape, train_y.shape, test_y.shape
```

```
((4704, 562), (1177, 562), (4704,), (1177,))
```

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier() # 하이파라미터 기본값으로 해보라는 것 같다...
rf.fit(train_X, train_y)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

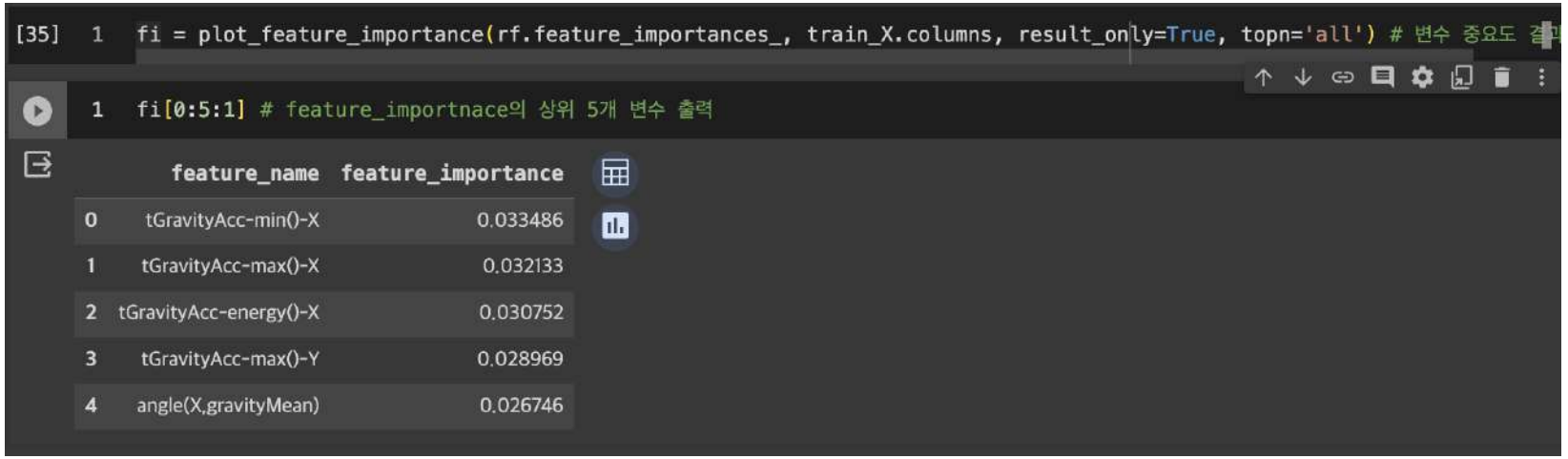
```
[ ]  print('훈련 세트 점수 : ', rf.score(train_X, train_y))
     print('평가 세트 점수 : ', rf.score(test_X, test_y))
```

```
훈련 세트 점수 :  1.0
평가 세트 점수 :  0.9804587935429057
```

6. 탐색적 데이터 분석

- 6개에 대해서 다중 분류했을 때 상위 5개 변수를 확인한다. (관점 1)

  -> 해당 5개 변수를 저장한다.



```
[35]  1  fi = plot_feature_importance(rf.feature_importances_, train_X.columns, result_only=True, topn='all') # 변수 중요도 결
```

```
1  fi[0:5:1] # feature_importnace의 상위 5개 변수 출력
```

	feature_name	feature_importance
0	tGravityAcc-min()-X	0.033486
1	tGravityAcc-max()-X	0.032133
2	tGravityAcc-energy()-X	0.030752
3	tGravityAcc-max()-Y	0.028969
4	angle(X,gravityMean)	0.026746

6. 탐색적 데이터 분석

- 정적 행동(0)과 동적 행동(1)를 이진 분류했을 때 상위 5개 변수를 확인한다. (관점 2)

```
# data를 copy해서 data2로 명명했다.

# STANDING, SITTING, LAYING은 정적 행동으로 0을 의미
# WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS은 동적 행동으로 1을 의미

# 따라서 이를 바타응로 'is_dynamic' 변수에 값을 추가한다.
data2['is_dynamic'] = data2['Activity'].apply(lambda x: 0 if x in ["STANDING", "SITTING", "LAYING"] else 1)
```

```
[ ]
# train과 test로 데이터 분할
X = data2.drop(['is_dynamic'], axis=1)
y = data2['is_dynamic']

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, random_state=42) # 데이터 불균형이 아닌 것 같아서
```

6. 탐색적 데이터 분석

- 정적 행동(0)과 동적 행동(1)를 이진 분류했을 때 상위 5개 변수를 확인한다. (관점 2)

  -> 해당 5개 변수를 저장한다.

```
1  rf2 = RandomForestClassifier() # 하이파라미터 기본값으로 해보라는 것 같다...
2  rf2.fit(train_X, train_y)
```

```
[45]  1  fi2 = plot_feature_importance(rf2.feature_importances_, train_X.columns, result_only=True, topn=5)
```

```
1  fi2
```

	feature_name	feature_importance
0	tBodyGyroJerk-sma()	0.049709
1	tBodyAccJerk-energy()-X	0.040076
2	tBodyAccJerk-mad()-X	0.039949
3	tBodyAccJerkMag-entropy()	0.039769
4	fBodyAccJerk-bandsEnergy()-1,16	0.030163

6. 탐색적 데이터 분석

- **‘STANDING’**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 3)

```
# data를 copy 해서 data3으로 명명한다.

# is_standing 변수를 추가
data3['is_standing'] = data3['Activity'].apply(lambda x : 1 if x == 'STANDING' else 0)
```

```
from itertools import starmap
# train과 test로 분할한다.
X = data3.drop(['is_standing'], axis=1)
y = data3['is_standing']

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

6. 탐색적 데이터 분석

- **'STANDING'**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 3)

  -> 해당 5개 변수를 저장한다.

```
1  rf3 = RandomForestClassifier()
2  rf3.fit(train_X, train_y)
```

```
[54]  1  fi3 = plot_feature_importance(rf3.feature_importances_, train_X.columns, result_only=True, topn=5)
```

```
1  fi3
```

	feature_name	feature_importance
0	angle(Y,gravityMean)	0.063043
1	tGravityAcc-min()-Y	0.049856
2	tGravityAcc-mean()-Y	0.046058
3	tGravityAcc-max()-Y	0.045271
4	tGravityAcc-max()-Z	0.025018

6. 탐색적 데이터 분석

- **‘SITTING’**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 4)

```
[57]  1  # data를 copy하여 data4로 명명한다.
      2
      3  data4['is_sitting'] = data4['Activity'].apply(lambda x : 1 if x == 'SITTING' else 0)
```

```
[60]  1  # train과 test로 분할
      2  X = data4.drop(['is_sitting'], axis=1)
      3  y = data4['is_sitting']
      4
      5  train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

6. 탐색적 데이터 분석

- **‘SITTING’**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 4)

  -> 해당 5개 변수를 저장한다.

```
1  rf4 = RandomForestClassifier()
2  rf4.fit(train_X, train_y)
```

```
[63]  1  fi4 = plot_feature_importance(rf4.feature_importances_, train_X.columns, result_only=True, topn=5)
```

```
1  fi4
```

	feature_name	feature_importance
0	angle(Y,gravityMean)	0.049390
1	tGravityAcc-mean()-Y	0.047824
2	tGravityAcc-max()-Y	0.035859
3	tGravityAcc-energy()-X	0.033640
4	tGravityAcc-min()-Y	0.031424

6. 탐색적 데이터 분석

- **‘LAYING’**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 5)

```
[66]  1  data5['is_laying'] = data5['Activity'].apply(lambda x : 1 if x == 'LAYING' else 0)
```

```
1  # train과 test 분할한다
2  X = data5.drop(['is_laying'], axis=1)
3  y = data5['is_laying']
4
5  train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

6. 탐색적 데이터 분석

‘LAYING’를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 5)
-> 해당 5개 변수를 저장한다.

```
1  rf5 = RandomForestClassifier()
2  rf5.fit(train_X, train_y)
```

```
[72]  1  fi5 = plot_feature_importance(rf5.feature_importances_, train_X.columns, result_only=True, topn=5)
```

```
1  fi5
```

	feature_name	feature_importance
0	tGravityAcc-min()-X	0.096832
1	tGravityAcc-max()-Y	0.094947
2	tGravityAcc-mean()-Y	0.077721
3	tGravityAcc-max()-X	0.077085
4	tGravityAcc-energy()-X	0.074728

6. 탐색적 데이터 분석

- **‘WALKING’**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 6)

```
# data copy해서 data6으로 명명한다.

# 'is_walking' 변수 추가
data6['is_walking'] = data['Activity'].apply(lambda x : 1 if x == 'WALKING' else 0)
```

```
# train과 test로 분할
X = data6.drop(['is_walking'], axis=1)
y = data6['is_walking']

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

6. 탐색적 데이터 분석

- **'WALKING'**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 6)

  -> 해당 5개 변수를 저장한다.

```
1  rf6 = RandomForestClassifier()
2  rf6.fit(train_X, train_y)
```

```
[81]  1  fi6 = plot_feature_importance(rf6.feature_importances_, train_X.columns, result_only=True, topn=5)
```

```
1  fi6
```

	feature_name	feature_importance
0	tGravityAcc-arCoeff()-X,1	0.023624
1	fBodyAccMag-mad()	0.023542
2	fBodyAccMag-std()	0.021143
3	tBodyAcc-correlation()-X,Y	0.019713
4	fBodyAcc-bandsEnergy()-1,8	0.019332

6. 탐색적 데이터 분석

‘WALKING_UP’를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 7)

```
# data를 copy해서 data7로 명명한다.

data7['is_walking_up'] = data7['Activity'].apply(lambda x : 1 if x == 'WALKING_UPSTAIRS' else 0)
```

```
# train과 test로 분할한다.
X = data7.drop(['is_walking_up'], axis=1)
y = data7['is_walking_up']

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

6. 탐색적 데이터 분석

- **'WALKING_UP'**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 7)

  -> 상위 5개 변수를 저장한다.

```
[88]  1  rf7 = RandomForestClassifier()
      2  rf7.fit(train_X, train_y)
```

```
[90]  1  fi7 = plot_feature_importance(rf7.feature_importances_, train_X.columns, result_only=True, topn=5)
```

```
1  fi7
```

	feature_name	feature_importance
0	tGravityAcc-arCoeff()-Z,2	0.040428
1	tGravityAcc-arCoeff()-Y,2	0.031420
2	tGravityAcc-min()-Y	0.031054
3	angle(Y,gravityMean)	0.029645
4	tGravityAcc-arCoeff()-Z,1	0.026457

6. 탐색적 데이터 분석

- **‘WALKING_DOWN’**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 8)

```
# data를 copy해서 data8로 명명한다.

# 'is_walking' 변수 추가
data8['is_walking_down'] = data8['Activity'].apply(lambda x : 1 if x == 'WALKING_DOWNSTAIRS' else 0)
```

```
[198]
# train, test 분할
X = data8.drop(['is_walking_down'], axis=1)
y = data8['is_walking_down']

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

6. 탐색적 데이터 분석

- **‘WALKING_DOWN’**를 1로, 나머지를 0으로 정의했을 때 상위 5개 변수를 확인한다. (관점 8)

```
1  rf8 = RandomForestClassifier()
2  rf8.fit(train_X, train_y)
```

```
[200] 1  fi8 = plot_feature_importance(rf8.feature_importances_, train_X.columns, result_only=True, topn=5)
```

```
1  fi8
```

	feature_name	feature_importance
0	tBodyAccMag-std()	0.076139
1	fBodyAccMag-mad()	0.041799
2	tBodyAcc-max()-X	0.039772
3	tBodyAcc-std()-X	0.034167
4	tGravityAccMag-std()	0.033410

6. 탐색적 데이터 분석

- **관점 1, 관점 2 ... 관점 8에서 각각 상위 5개 변수들을 모아 새로운 변수 vv에 저장한다.**
  -> 기본 모델링, 단계별 모델링할 때 이런 변수들이 활용될 예정이다.

```
vv = list(set(fi[0:5:1]['feature_name'].tolist() + \
              fi2['feature_name'].tolist() +\
              fi3['feature_name'].tolist() +\
              fi4['feature_name'].tolist() +\
              fi5['feature_name'].tolist() +\
              fi6['feature_name'].tolist() +\
              fi7['feature_name'].tolist() +\
              fi8['feature_name'].tolist()))
```

```
vv # 리스트 출력
```

```
['tBodyAcc-std()-X',
 'tGravityAcc-max()-Y',
 'tBodyAccJerkMag-entropy()',
 'fBodyAcc-max()-X',
 'tGravityAccMag-std()',
 'tGravityAcc-min()-Y',
 'tGravityAcc-mean()-Y',
 'tGravityAcc-mean()-X',
 'tGravityAccMag-arCoeff()1',
 'tBodyAccJerk-mad()-X',
 'fBodyAccMag-mad()',
 'fBodyAcc-bandsEnergy()-1,8.1',
 'fBodyAccJerk-bandsEnergy()-1,8',
 'tGravityAcc-min()-X',
 'tGravityAcc-mean()-Z',
 'fBodyAccJerk-std()-Y',
 'tBodyAcc-max()-X',
 'tBodyGyroJerk-iqr()-X',
 'tGravityAcc-max()-X',
 'tGravityAcc-energy()-X',
 'fBodyAcc-meanFreq()-Z',
 'tGravityAcc-arCoeff()-Z,1',
 'tBodyAccMag-std()',
 'fBodyAccJerk-bandsEnergy()-1,16',
 'angle(Y,gravityMean)',
 'fBodyAccJerk-bandsEnergy()-17,24.2']
```

7. 기본 모델링

- 최적의 변수를 이용해 기본 모델링을 수행하기 위한 사전 작업을 한다.

```
1  vv # 최적의 변수 출력
```

```
['tBodyAcc-energy()-X',
 'angle(X,gravityMean)',
 'tBodyGyroJerk-iqr()-Z',
 'tGravityAcc-arCoeff()-Z,3',
 'tGravityAcc-min()-X',
 'tGravityAcc-max()-Y',
 'tBodyAccMag-std()',
 'tBodyGyroJerk-mad()-X',
 'fBodyAccMag-sma()',
 'tGravityAcc-mean()-Z',
 'tGravityAcc-min()-Y',
 'fBodyAcc-mad()-X',
 'tGravityAcc-mean()-Y',
 'angle(Y,gravityMean)',
 'fBodyAccMag-std()',
 'tGravityAccMag-std()',
 'tGravityAcc-mean()-X',
 'tBodyAccJerk-mad()-X',
 'fBodyAcc-energy()-X',
 'tBodyAcc-max()-X',
 'tBodyAccJerk-mad()-Y',
 'tGravityAcc-arCoeff()-X,1',
 'tGravityAcc-energy()-X',
 'tGravityAcc-max()-X',
 'fBodyAcc-meanFreq()-Z',
 'tBodyAccJerk-iqr()-Y',
 'tGravityAcc-arCoeff()-Z,2']
```

```
1  # data를 copy해서 data2로 명명한다.
2
3  data2 = data2[vv + ['Activity']] # 해당 변수만 남겨둔다.
```

```
1  data2.shape
```

```
(5881, 28)
```

7. 기본 모델링

- 최적의 변수를 이용해 기본 모델링을 수행하기 위한 사전 작업을 한다.
  -> Train 셋과 Test 셋을 준비한다.

```
[12]  1  x2 = data2.drop(['Activity'], axis=1)
      2  y2 = data2['Activity']
```

```
[14]  1  train_X2, test_X2, train_y2, test_y2 = train_test_split(x2, y2, test_size=0.2, random_state=42)
```

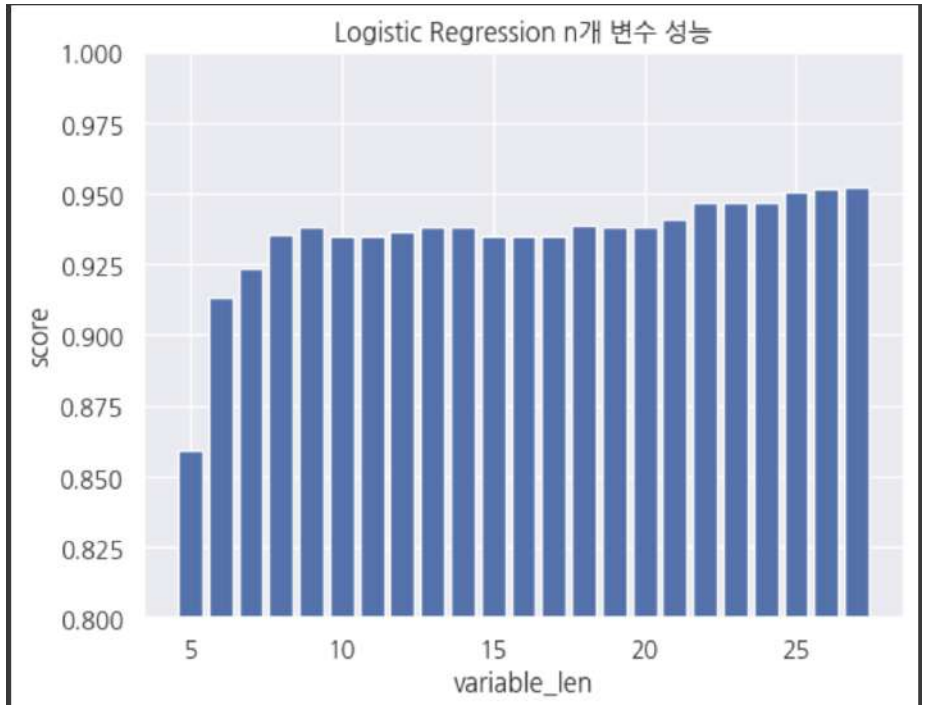
```
[15]  1  print(train_X2.shape, test_X2.shape, train_y2.shape, test_y2.shape)
```

```
(4704, 27) (1177, 27) (4704,) (1177,)
```

7. 기본 모델링

- **5개 변수 ~ 27개 변수까지 하나씩 해서 모델의 정확도를 확인한다.**
  -> Logistic Regression

```
[22]  1  variable1_len = []
      2  score1 = []
      3  for i in range(5, 28, 1):
      4      temp_train_X2 = train_X2[train_X2.columns[0:i:1]]
      5      temp_test_X2 = test_X2[test_X2.columns[0:i:1]]
      6
      7      lr2 = LogisticRegression()
      8      lr2.fit(temp_train_X2, train_y2)
      9
     10      print(f'변수 {i}개일 때')
     11      print(f'평가 세트 점수 : {lr2.score(temp_test_X2, test_y2)}')
     12
     13      variable1_len.append(i)
     14      score1.append(lr2.score(temp_test_X2, test_y2))
```



7. 기본 모델링

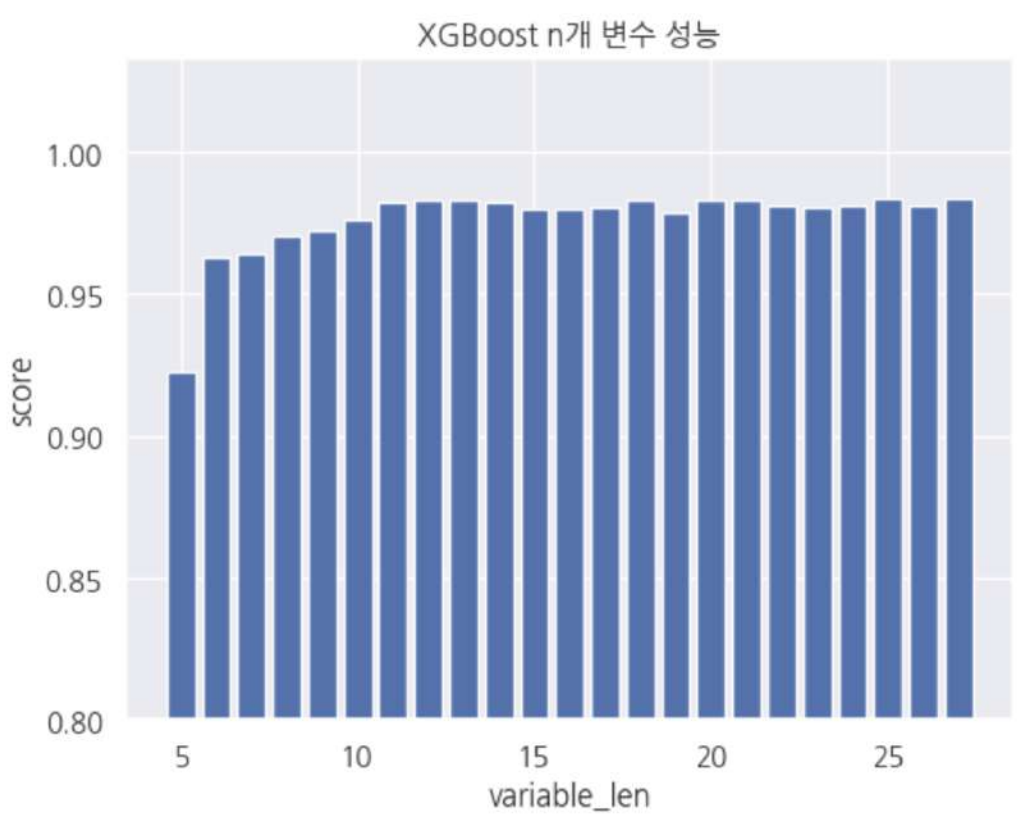
- **5개 변수 ~ 27개 변수까지 하나씩 해서 모델의 정확도를 확인한다.**
  -> Decision Tree

```
[67]  1  variable2_len = []
      2  score2 = []
      3  for i in range(5, 28, 1):
      4      temp_train_X2 = train_X2[train_X2.columns[0:i:1]]
      5      temp_test_X2 = test_X2[test_X2.columns[0:i:1]]
      6
      7      dt2 = DecisionTreeClassifier()
      8      dt2.fit(temp_train_X2, train_y2)
      9
     10      print(f'변수 {i}개일 때')
     11      print(f'평가 세트 점수 : {dt2.score(temp_test_X2, test_y2)}')
     12
     13      variable2_len.append(i)
     14      score2.append(dt2.score(temp_test_X2, test_y2))
```

7. 기본 모델링

- **5개 변수 ~ 27개 변수까지 하나씩 해서 모델의 정확도를 확인한다.**
  -> XGBoost

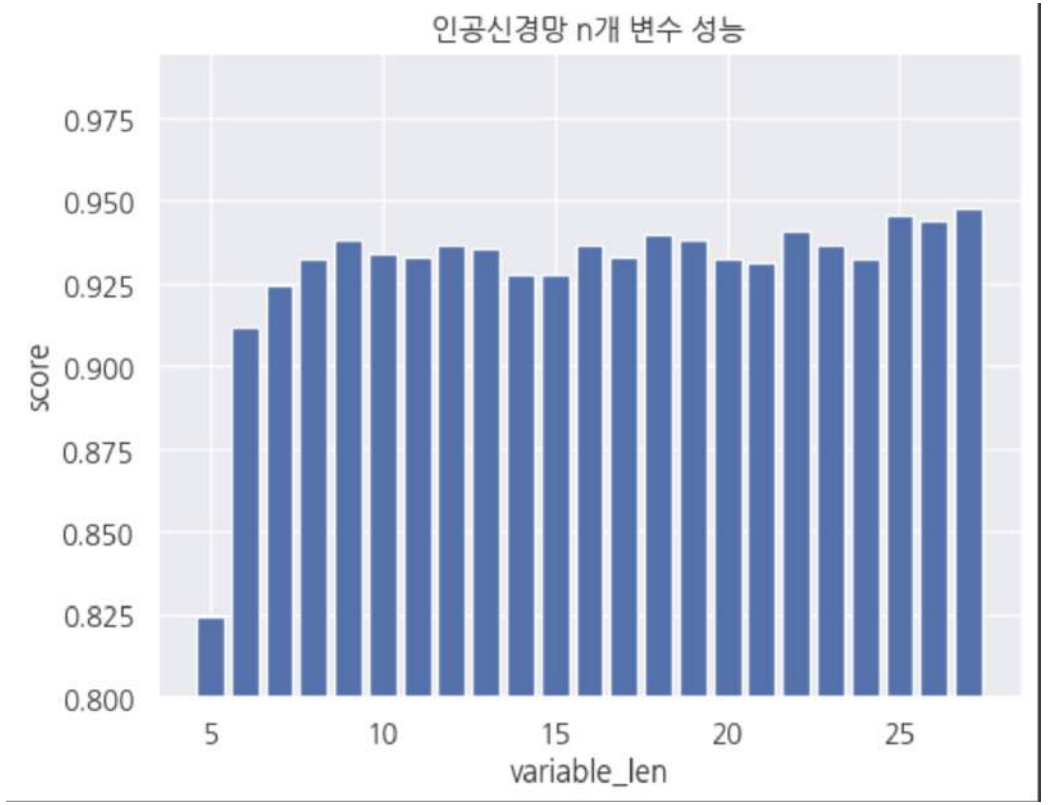
```python
variable3_len = []
score3= []

for i in range(5, 28, 1):
    temp_train_X2 = train_X2[train_X2.columns[0:i:1]]
    temp_test_X2 = test_X2[test_X2.columns[0:i:1]]

    # 레이블 인코더 생성
    le2 = LabelEncoder()

    # 학습 데이터의 레이블을 학습하고 변환
    train_y2_encoded = le2.fit_transform(train_y2)

    # 테스트 데이터의 레이블을 변환 (필요한 경우)
    test_y2_encoded = le2.transform(test_y2)

    # XGBoost 분류기 생성
    xgb2 = xgb.XGBClassifier()

    # 인코딩된 레이블을 사용하여 모델 학습
    xgb2.fit(temp_train_X2, train_y2_encoded)

    print(f'변수 {i}개일 때')
    print(f'평가 세트 점수 : {xgb2.score(temp_test_X2, test_y2_encoded)}')

    variable3_len.append(i)
    score3.append(xgb2.score(temp_test_X2, test_y2_encoded))
```



7. 기본 모델링

- **5개 변수 ~ 27개 변수까지 하나씩 해서 모델의 정확도를 확인한다.**
  -> 인공신경망

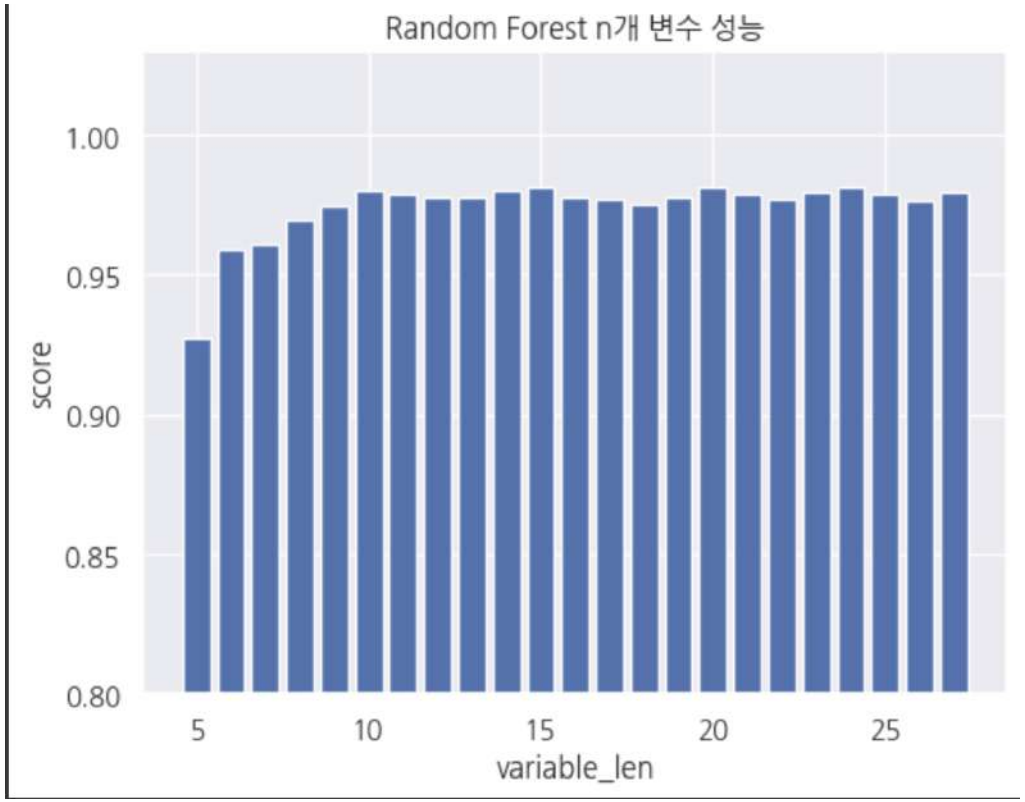
```python
variable4_len = []
score4 = []
for i in range(5, 28, 1):
    temp_train_X2 = train_X2[train_X2.columns[0:i:1]]
    temp_test_X2 = test_X2[test_X2.columns[0:i:1]]

    # 레이블 인코더 생성
    dd2 = LabelEncoder()

    # 학습 데이터의 레이블을 학습하고 변환
    train_y2_encoded = dd2.fit_transform(train_y2)

    # 테스트 데이터의 레이블을 변환 (필요한 경우)
    test_y2_encoded = dd2.transform(test_y2)

    # 딥러닝 구조 설계
    model2 = Sequential()
    model2.add(Dense(128, activation='swish', input_shape=(temp_train_X2.shape[1], )))
    model2.add(Dropout(0.2))
    model2.add(Dense(64, activation='swish'))
    model2.add(Dropout(0.2))
    model2.add(Dense(32, activation='swish'))
    model2.add(Dropout(0.2))
    model2.add(Dense(6, activation='softmax'))

    model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# 딥러닝 모델 학습
model2.fit(temp_train_X2, train_y2_encoded, epochs=10, batch_size=32, validation_split=0.2)

# 예측값 추출하기
y_pred = np.argmax(model2.predict(temp_test_X2), axis=1)

print(f'변수 {i}개일 때')
print('accuracy_score : ', accuracy_score(test_y2_encoded, y_pred)) # 실제값과 예측값 비교를 통해 정확도 도출

variable4_len.append(i)
score4.append(accuracy_score(test_y2_encoded, y_pred))
```



7. 기본 모델링

- **5개 변수 ~ 27개 변수까지 하나씩 해서 모델의 정확도를 확인한다.**
  -> Random Forest

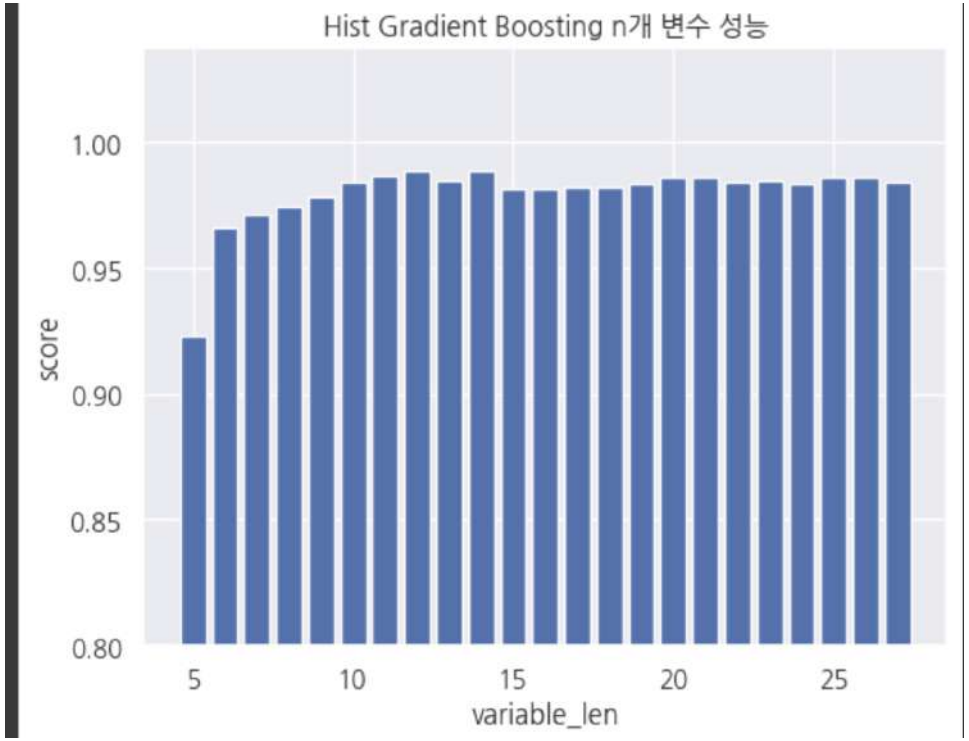
```python
variable5_len = []
score5 = []
for i in range(5, 28, 1):
    temp_train_X2 = train_X2[train_X2.columns[0:i:1]]
    temp_test_X2 = test_X2[test_X2.columns[0:i:1]]

    rf = RandomForestClassifier()
    rf.fit(temp_train_X2, train_y2)

    print(f'변수 {i}개일 때')
    print(f'평가 세트 점수 : {rf.score(temp_test_X2, test_y2)}')

    variable5_len.append(i)
    score5.append(rf.score(temp_test_X2, test_y2))
```



7. 기본 모델링

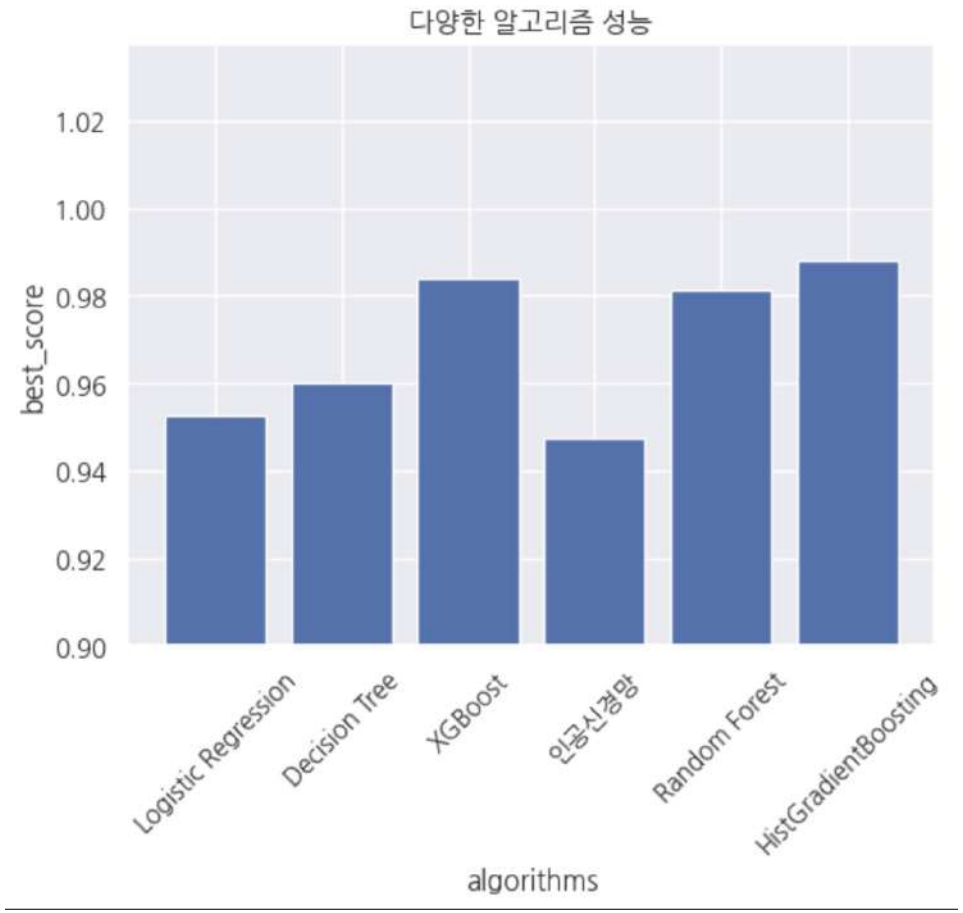
- **5개 변수 ~ 27개 변수까지 하나씩 해서 모델의 정확도를 확인한다.**
  -> Hist Gradient Boosting

```python
variable6_len = []
score6 = []
for i in range(5, 28, 1):
    temp_train_X2 = train_X2[train_X2.columns[0:i:1]]
    temp_test_X2 = test_X2[test_X2.columns[0:i:1]]

    # 모델 생성 및 훈련
    hgb = HistGradientBoostingClassifier(random_state=42)
    hgb.fit(temp_train_X2, train_y2)

    print(f'변수 {i}개일 때')
    print(f'평가 세트 점수 : {hgb.score(temp_test_X2, test_y2)}')

    variable6_len.append(i)
    score6.append(hgb.score(temp_test_X2, test_y2))
```



7. 기본 모델링

다양한 알고리즘의 성능을 전반적으로 확인한다.



8. 단계별 모델링

- 변수 12, 14개를 사용하면서 Hist Gradient Boosting 알고리즘을 사용했을 때 최대 성능이 나왔다.
- 따라서 변수 12개를 사용하고 Hist Gradient Boosting 알고리즘을 사용하면서 단계별 모델링을 수행하고자 한다.

8. 단계별 모델링

단계별 모델링을 수행하기 위한 사전 작업을 수행한다.
-> 변수 12개만 사용한다.

```
[57]  1  data2 = data[vv[0:12:1] + ['Activity']]
```

```
1  data2.shape
```

```
(5881, 13)
```

8. 단계별 모델링

정적 행동(LAYING, SITTING, STANDING)이면 0를 부여
동적 행동(WALKING, WALKING-UP, WALKING-DOWN이면 1를 부여
-> 모델 1에 활용할 예정이다.

```
# 'Activity_dynamic' 변수 추가
# '정적 행동(Laying, Sitting, Standing)'이면 0을 부여
# '동적 행동(Walking, Walking-Up, Walking-Down)'이면 1을 부여
data2['Activity_dynamic'] = data2['Activity'].apply(lambda x : 1 if x in ['WALKING', 'WALKING_UPSTAIRS', 'WALKING_D
```

```
data2['Activity_dynamic'].value_counts() # 이정도면 클래스 불균형이 있지는 않다...
```

```
0    3234
1    2647
Name: Activity_dynamic, dtype: int64
```

8. 단계별 모델링

Activity 변수를 문자열 형태에서 숫자 형태로 변환

-> 일부 머신러닝 알고리즘은 문자열을 자동적으로 0부터 연속하는 정수로 변환하지만,
그렇지 않은 알고리즘도 존재한다. 따라서 마음 편하게 숫자 형태로 변환하는 것이 좋다.

```
# 'Activity' 변수에 있는 값들을 숫자 형태로 변경한다.
data2['Activity'] = data2['Activity'].map({
    'STANDING' : 0,
    'LAYING' : 1,
    'SITTING' : 2,
    'WALKING' : 3,
    'WALKING_DOWNSTAIRS' : 4,
    'WALKING_UPSTAIRS' : 5,
})
```

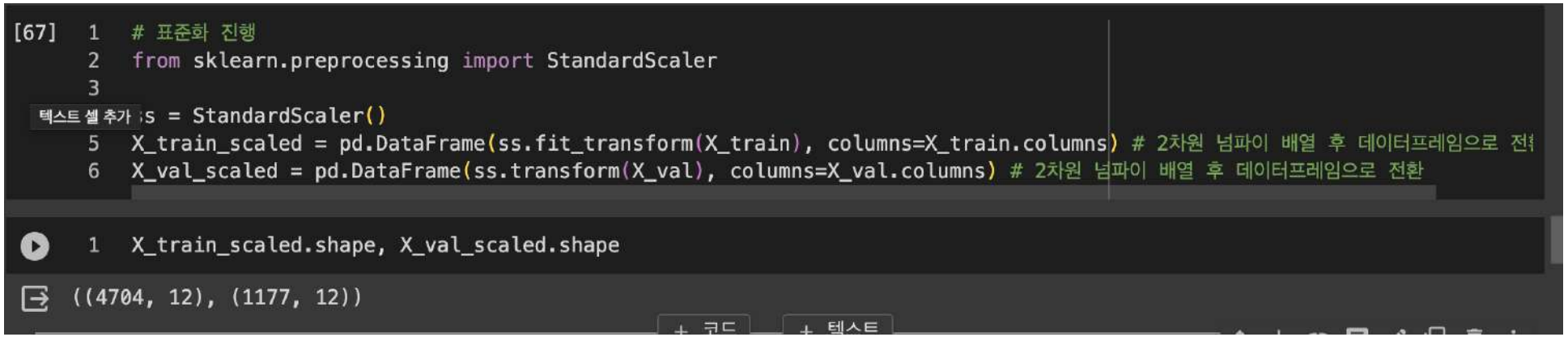
8. 단계별 모델링

단계별 모델링을 수행하기 위해 훈련 세트와 검증 세트로 분할한다.

```
# 입력 변수 X와 타겟 변수 y1, y2 분리
X = data2.drop(columns=['Activity', 'Activity_dynamic'], axis=1)
y1 = data2['Activity']
y2 = data2['Activity_dynamic']

# 훈련 세트와 검증 세트 분할 (train : val = 8 : 2)
X_train, X_val, y1_train, y1_val, y2_train, y2_val = train_test_split(X, y1, y2, test_size=0.2, random_state=42)
```

8. 단계별 모델링

표준화 진행



8. 단계별 모델링

정적 행동(0)과 동적 행동(1)를 구분하는 모델 1을 학습하고, 검증용 데이터로 예측값을 추출한다.

```
1  # 정적 행동(0)과 동적 행동(1)을 구분하는 모델
2  model1 = HistGradientBoostingClassifier(random_state=42, )
3  model1.fit(X_train_scaled, y2_train)
```

```
[71]  1  model1_predict= model1.predict(X_val_scaled) # 예측값이 0이 나왔는지 1이 나왔는지 확인
```

```
1  model1_predict # 예측값은 0(정적 행동), 1(동적 행동)로 구성된다.
```

```
array([0, 1, 0, ..., 0, 1, 0])
```

8. 단계별 모델링

정적 행동을 0, 1, 2로 분류하는 모델 2-1를 만들기 위한 사전 작업이다.
-> 해당 클래스(0, 1, 2)는 데이터 균형인 것 같아서 별다른 조치는 하지 않았다.

```
[73]  1  static_index = y1_train[y1_train.isin([0, 1, 2])].index # y1_train에서 0, 1, 2만 있는 index를 추출한다.
      2  static_X_train_scaled = X_train_scaled.loc[static_index] # 해당된 index에만 적용
      3  static_y_train = y1_train.loc[static_index] # 해당된 index에만 적용
```

```
  1  static_y_train.value_counts()
```

```
1    884
0    861
2    832
Name: Activity, dtype: int64
```

8. 단계별 모델링

정적 행동을 분류하는 모델 2-1를 학습한다.

```
# 정적 행동을 분류하는(0, 1, 2) 모델 2-1
model2_1 = HistGradientBoostingClassifier(random_state=42)
model2_1.fit(static_X_train_scaled, static_y_train)
```

8. 단계별 모델링

동적 행동을 3, 4, 5로 분류하는 모델 2-2를 만들기 위한 사전 작업이다.

-> 해당 클래스(3, 4, 5)는 데이터 균형인 것 같아서 별다른 조치는 하지 않았다.

```
[78]  1  dynamic_index = y1_train[y1_train.isin([3, 4, 5])].index # y1_train에서 3, 4, 5만 있는 index를 추출한다.
      2  dynamic_X_train_scaled = X_train_scaled.loc[dynamic_index] # 해당된 index에만 적용
      3  dynamic_y_train = y1_train.loc[dynamic_index] # 해당된 index에만 적용
```

```
[80]  1  dynamic_y_train.value_counts()

     3    800
     5    681
     4    646
     Name: Activity, dtype: int64
```

8. 단계별 모델링

동적 행동을 분류하는 모델 2-2를 학습한다.

```
# 동적 행동을 분류하는(3, 4, 5) 모델 2-2
model2_2 = HistGradientBoostingClassifier(random_state=42)
model2_2.fit(dynamic_X_train_scaled, dynamic_y_train)
```

8. 단계별 모델링

검증용 데이터로 모델 1를 거쳐 모델 2-1 혹은 모델 2-2를 수행하는 파이프라인을 거쳐서 예측값을 저장한다.

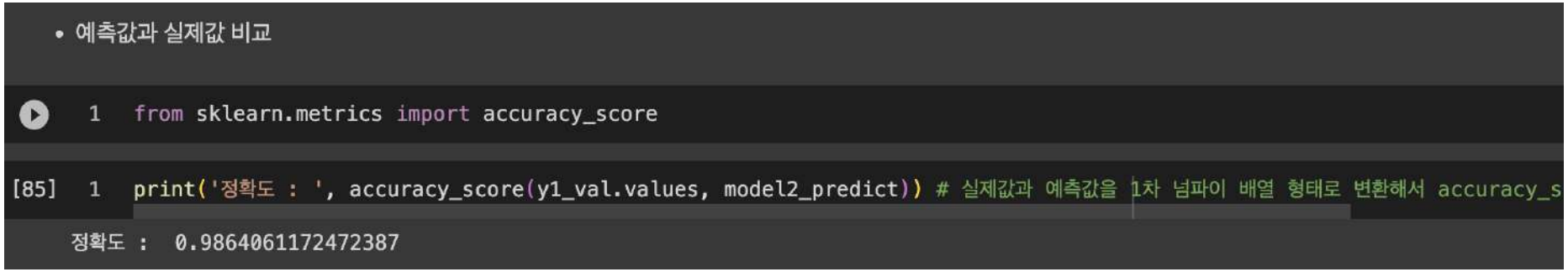
```
[77]  1  # 모델 2-1 Laying(0), Sitting(1), Standing(2)를 다중 분류하는 함수
      2  def static_behavior_classification(idx):
      3      result = model2_1.predict(X_val_scaled.iloc[idx].values.reshape(1, -1)) # Series를 2차원 넘파이 배열로 왜냐하면 머신러닝,
      4      return result[0]
```

```
      1  # 모델 2-2 Walking(3), Walking-Up(4), Walking-Down(5)를 다중 분류하는 함수
      2  def dynamic_behavior_classification(idx):
      3      result = model2_2.predict(X_val_scaled.iloc[idx].values.reshape(1, -1)) # Series를 2차원 넘파이 배열로 왜냐하면 머신
      4      return result[0]
```

```
      1  model2_predict = [] # model2_predict(예측값)는 y1_val(실제값)과 비교할 예정이다.
      2
      3  for idx, mp in enumerate(model1_predict): # model1_predict(예측값)을 이용해서 모델 2(모델 2-1, 모델 2-2)를 수행할 예정이다.
      4      if mp == 0:
      5          result = static_behavior_classification(idx)
      6          model2_predict.append(result)
      7      else: # mp == 1
      8          result = dynamic_behavior_classification(idx)
      9          model2_predict.append(result)
     10
     11  model2_predict = np.array(model2_predict) # 1차원 넘파이 배열로 변환
```

8. 단계별 모델링

실제값과 예측값을 비교하여 정확도를 추출한다.



8. 단계별 모델링

진정한 테스트 데이터를 이용하여 정확도를 추출했다.

```
[97]  1  print('정확도 : ', accuracy_score(y_true, model2_predict)) # 실제값과 예측값을 1차 넘파이 배열 형태로 변환해서 accuracy_score를 진
    정확도 :  0.9775662814411965
```