분류 분석 II

빅데이터 분석

센서 데이터로 움직임 분류하기

목표 : 스마트폰에서 수집한 센서 데이터를 분석하여 사람의 움직임을 분류하는 모델을 생성

새로운 데이터에 대해 **움직임 유형을 예측해서 분류**

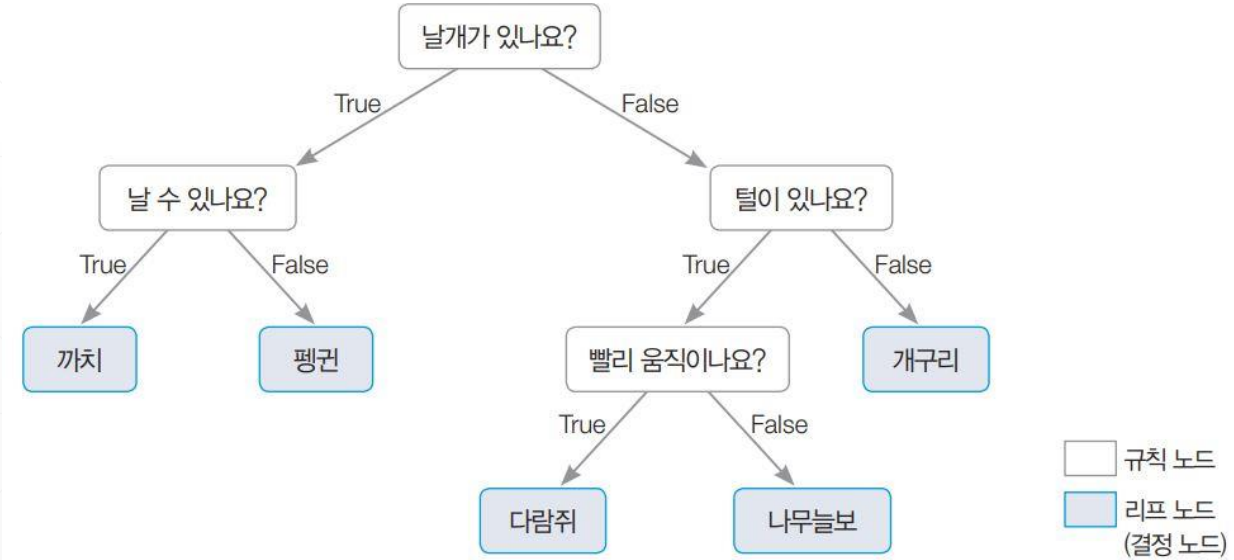


그림 11-4 {개구리, 펭귄, 까치, 나무늘보, 다람쥐}를 분류하기 위한 결정 트리

결정 트리 분석

- **DecisionTreeClassifier**
  - 사이킷런에서 제공하는 결정 트리 분류 모델

표 11-2 DecisionTreeClassifier의 주요 매개변수

매개변수	설명
min_samples_split	노드를 분할하기 위한 최소 샘플 데이터 개수(default: 2)
min_samples_leaf	리프 노드가 되기 위한 최소 샘플 데이터 개수
max_features	최적의 분할을 위해 고려할 최대 피처 개수 • None: 모든 피처 사용 • int: 사용할 피처 개수를 설정 • float: 사용할 피처 개수를 퍼센트로 설정 • sqrt: $\sqrt{(\text{전체 피처 개수})}$를 계산하여 설정 • auto: sqrt와 동일 • log: $\log_2(\text{전체 피처 개수})$를 계산하여 설정
max_depth	트리의 최대 깊이
max_leaf_nodes	리프 노드에 들어갈 수 있는 샘플 데이터의 최대 개수

- **Graphviz**
  - 패키지 결정 트리 시각화에 사용하는 패키지
  - 다이어그램을 그리기 위해 AT&T에서 개발한 그래프 시각화 오픈 소스 프로그램

붓꽃 데이터

Iris Versicolor

Iris Setosa

Iris Virginica

- **Sepal length** – 꽃받침의 길이 정보
- **Sepal width** – 꽃받침의 너비 정보
- **Petal length** – 꽃잎의 길이 정보
- **Petal width** – 꽃잎의 너비 정보
- Species (**Target**) – 꽃의 종류 정보 (**setosa / versicolor / virginica**)

데이터 탐색 및 결정 트리 학습

In [1]:	from sklearn.tree import DecisionTreeClassifier from sklearn.datasets import load_iris from sklearn.model_selection import train_test_split import warnings warnings.filterwarnings('ignore')

In [2]:	# 붓꽃 데이터를 로딩
	iris_data = load_iris()

In [3]:	iris_data.feature_names
In [4]:	iris_data.target_names

데이터 탐색

결정 트리 학습

In [5]:	# 학습과 테스트 데이터 셋으로 분리 X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data.target, test_size=0.2, random_state=11)

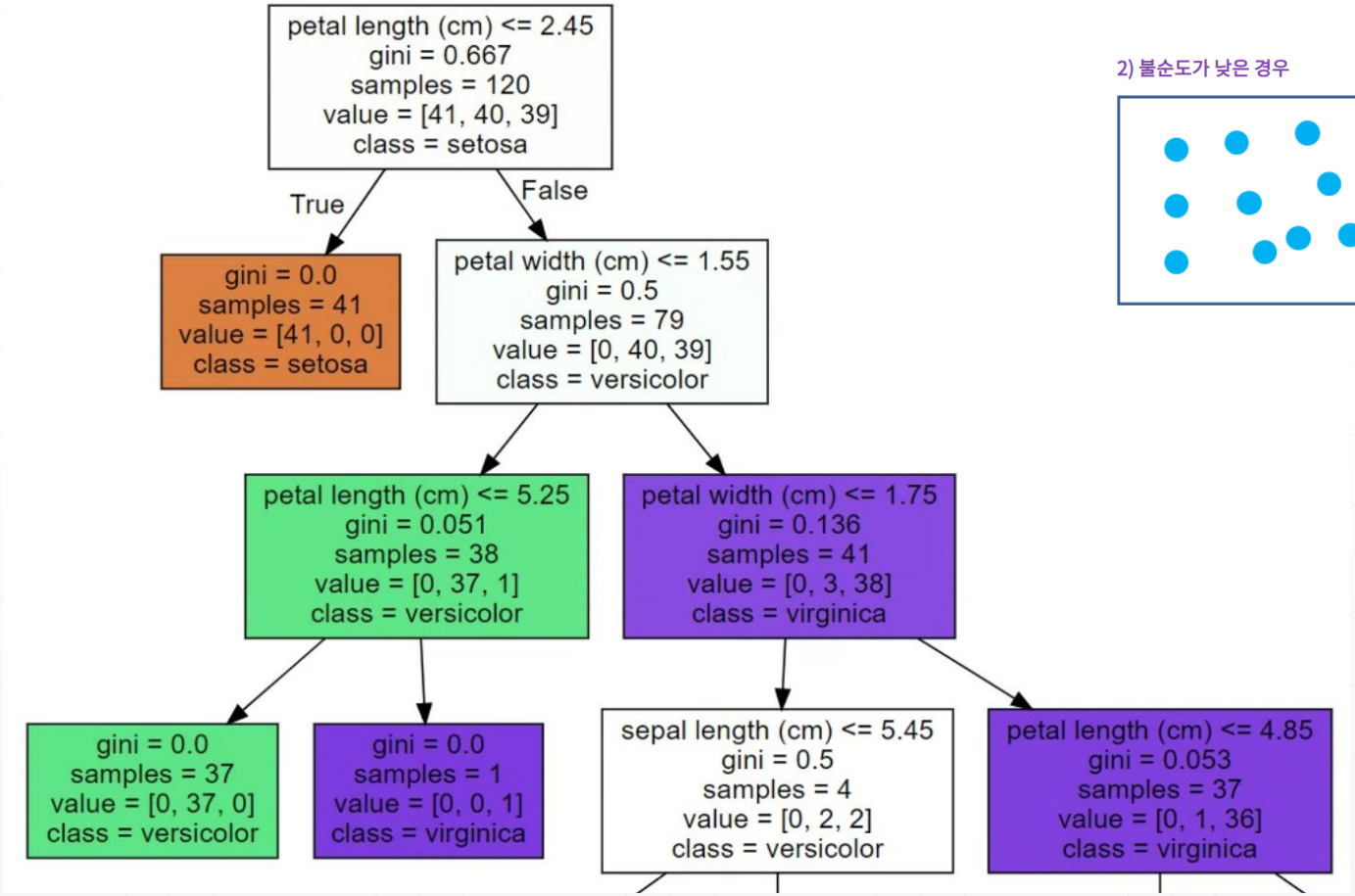
In [6]:	# DecisionTreeClassifier 생성
	dt_clf = DecisionTreeClassifier(random_state=156)

In [7]:	# DecisionTreeClassifer 학습
	dt_clf.fit(X_train, y_train)

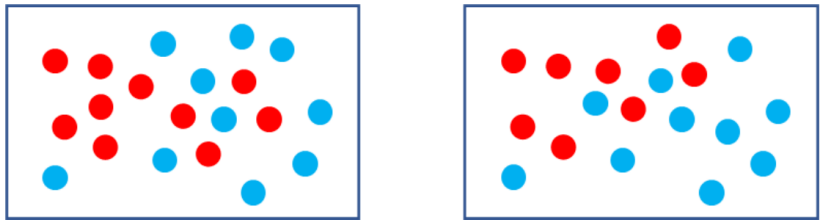
결정 트리 시각화

In [8]:	# https://www2.graphviz.org/Packages/stable/windows/10/cmake/Release/x64/ ##graphviz-install-2.44.1-win64.exe를 다운로드하여 설치하기 import os ###설치 경로(C:/Program Files/Graphviz 2.44.1/bin)를 PATH에 추가하기 os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz 2.44.1/bin'
In [9]:	from sklearn.tree import export_graphviz # export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일을 생성함 export_graphviz(dt_clf, out_file="tree.dot", class_names=iris_data.target_names, feature_names=iris_data.feature_names, impurity=True, filled=True)
In [10]:	import graphviz # 위에서 생성된 tree.dot 파일을 Graphviz 읽어서 시각화 with open("tree.dot") as f: dot_graph = f.read() graphviz.Source(dot_graph)

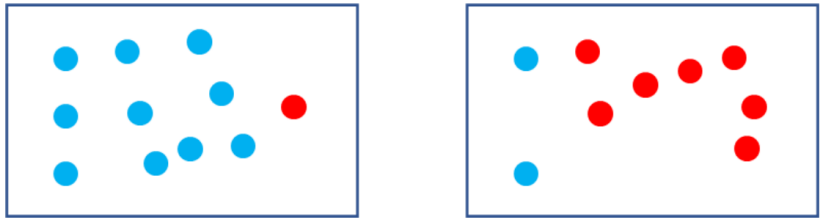
결정 트리 시각화 (cont'd)



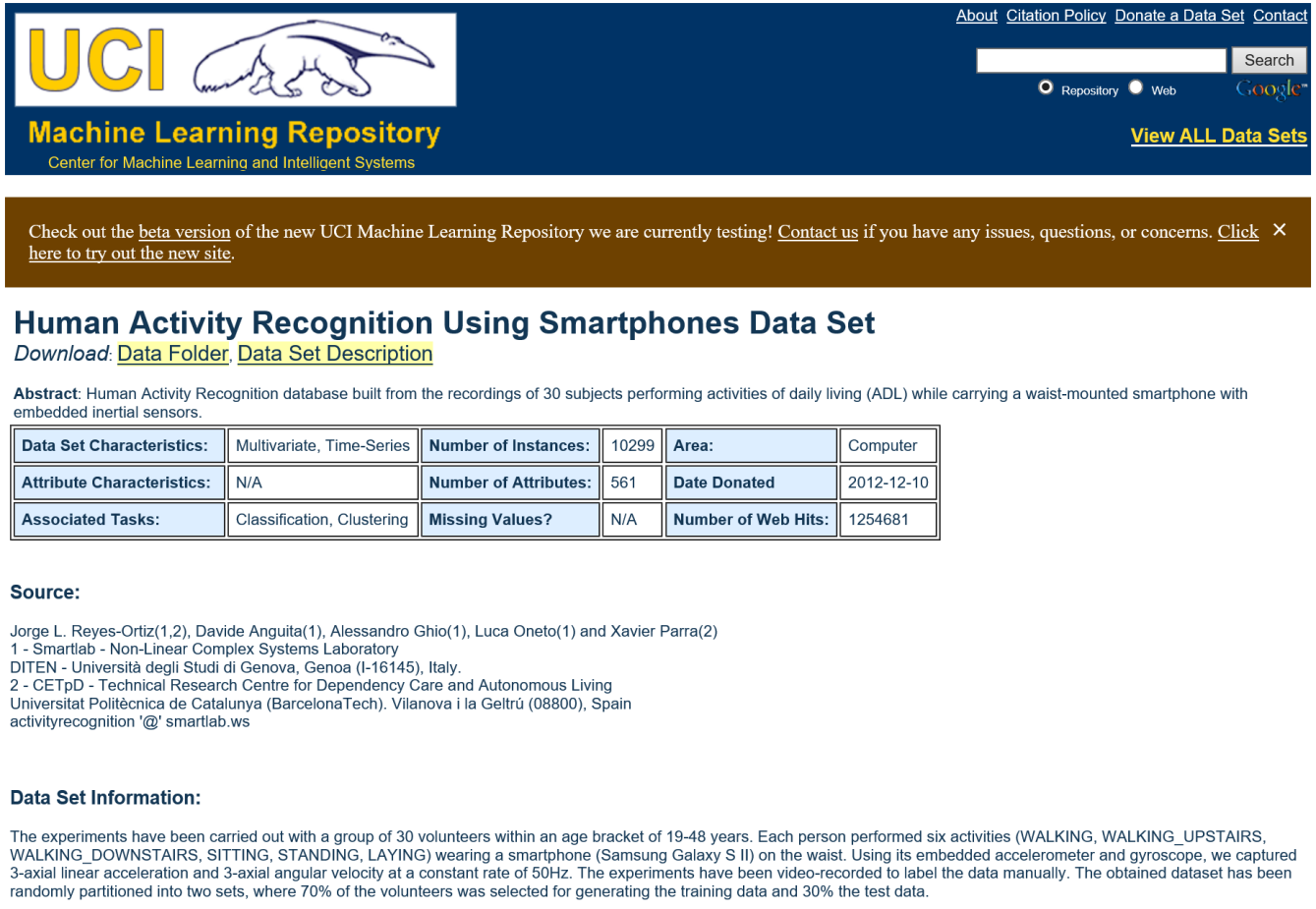
1) 불순도가 높은 경우



2) 불순도가 낮은 경우



센서 데이터로 움직임 분류하기



UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

About Citation Policy Donate a Data Set Contact

Search
Repository Web
Google

View ALL Data Sets

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:**	1254681

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

Data Set Information:

The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

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

센서 데이터로 움직임 분류하기 (cont'd)

1. 데이터 확인하기

In [1]:	import numpy as np import pandas as pd

In [2]:	# 피처 이름 파일 읽어오기
	feature_name_df = pd.read_csv('I:/My_Python/UCI_HAR_Dataset/features.txt', sep = '₩s+',
	header = None, names = ['index', 'feature_name'], engine = 'python')
	feature_name_df

In [3]:	feature_name_df.head()
In [4]:	feature_dup_df = feature_name_df.groupby('feature_name').count() feature_dup_df
In [5]:	feature_dup_df = feature_name_df.groupby('feature_name').count() print(feature_dup_df[feature_dup_df['index'] > 1].count()) feature_dup_df[feature_dup_df['index'] > 1].head(10)

센서 데이터로 움직임 분류하기 (cont'd)

2. 중복된 이름 해결

In [6]:

```
def get_new_feature_name_df(old_feature_name_df):
    feature_dup_df = pd.DataFrame(data=old_feature_name_df.groupby('feature_name').cumcount(), columns=['dup_cnt'])
    feature_dup_df = feature_dup_df.reset_index()
    new_feature_name_df = pd.merge(feature_name_df.drop(columns='index').reset_index(), feature_dup_df, how='outer')
    new_feature_name_df['feature_name'] = new_feature_name_df[['feature_name', 'dup_cnt']].apply(lambda x : x[0]+'_'+str(x[1])
                                                                                                if x[1]>0 else x[0],  axis=1)
    new_feature_name_df = new_feature_name_df.drop(['index'], axis=1)
    return new_feature_name_df
```

In [7]:

```
# 중복된 피처명을 수정하는 get_new_feature_name_df()를 이용, 신규 피처명 DataFrame 생성
new_feature_name_df = get_new_feature_name_df(feature_name_df)
new_feature_name_df
```

In [8]:

```
# DataFrame에 피처명을 컬럼으로 부여하기 위해 리스트 객체로 다시 변환
feature_name = new_feature_name_df.iloc[:, 0].values.tolist()
feature_name
```

센서 데이터로 움직임 분류하기 (cont'd)

3. 훈련용과 테스트용 데이터셋 확인하기

In [9]:	X_train = pd.read_csv('I:/My_Python/UCI_HAR_Dataset/train/X_train.txt', sep='₩s+', names = feature_name, engine = 'python') X_test = pd.read_csv('I:/My_Python/UCI_HAR_Dataset/test/X_test.txt', sep='₩s+', names = feature_name, engine = 'python') Y_train = pd.read_csv('I:/My_Python/UCI_HAR_Dataset/train/y_train.txt', sep='₩s+', header = None, names = ['action'], engine = 'python') Y_test = pd.read_csv('I:/My_Python/UCI_HAR_Dataset/test/y_test.txt' , sep = '₩s+', header = None, names = ['action'], engine = 'python')
In [10]:	X_train.shape, Y_train.shape, X_test.shape, Y_test.shape
In [11]:	X_train.head()
In [12]:	print(Y_train['action'].value_counts())
In [13]:	label_name_df = pd.read_csv('I:/My_Python/UCI_HAR_Dataset/activity_labels.txt', sep = '₩s+', header = None, names = ['index', 'label'], engine = 'python')

In [14]:	# index 제거하고, feature_name만 리스트로 저장
	label_name = label_name_df.iloc[:, 1].values.tolist()

In [15]:	label_name

센서 데이터로 움직임 분류하기 (cont'd)

4. 결정 트리 분류 분석 모델 구축하기

In [16]:	from sklearn.tree import DecisionTreeClassifier

In [17]:	# 결정 트리 분류 분석: 모델 생성
	dt_HAR = DecisionTreeClassifier(random_state=156)

In [18]:	# 결정 트리 분류 분석: 모델 훈련
	dt_HAR.fit(X_train, Y_train)

In [19]:	# 결정 트리 분류 분석: 평가 데이터에 예측 수행 -> 예측 결과로 Y_predict 구하기
	Y_predict = dt_HAR.predict(X_test)

센서 데이터로 움직임 분류하기 (cont'd)

5. 생성한 모델의 성능 확인

In [20]:	from sklearn.metrics import accuracy_score
In [21]:	accuracy = accuracy_score(Y_test, Y_predict) print('결정 트리 예측 정확도: {0:.4f}'.format(accuracy))
In [22]:	print('결정 트리의 현재 하이퍼 매개변수: ₩n', dt_HAR.get_params())

GridSearchCV

6. 분류 정확도 높이기 I

	param_max_depth	mean_test_score	mean_train_score
0	6	0.850791	0.944879
1	8	0.851069	0.982692
2	10	0.851209	0.993403
3	12	0.844135	0.997212
4	16	0.851344	0.999660
5	20	0.850800	0.999966
6	24	0.849440	1.000000

In [23]:	from sklearn.model_selection import GridSearchCV

In [24]:	params = {
	'max_depth' : [6, 8, 10, 12, 16, 20, 24]
	}
	
	grid_cv = GridSearchCV(dt_HAR, param_grid = params, scoring = 'accuracy', cv = 5, return_train_score = True)
	
	grid_cv.fit(X_train, Y_train)

In [25]:	cv_results_df = pd.DataFrame(grid_cv.cv_results_) cv_results_df[['param_max_depth', 'mean_test_score', 'mean_train_score']]
In [26]:	print('최고 평균 정확도: {0:.4f}, 최적 하이퍼 매개변수: {1}'.format(grid_cv.best_score_, grid_cv.best_params_))

GridSearchCV (cont'd)

7. 분류 정확도 높이기 II

	param_max_depth	param_min_samples_split	mean_test_score	mean_train_score
0	8	8	0.852023	0.981468
1	8	16	0.854879	0.979836
2	8	24	0.851342	0.978237
3	16	8	0.844136	0.994457
4	16	16	0.847127	0.990479
5	16	24	0.849439	0.986772
6	20	8	0.846040	0.994491
7	20	16	0.848624	0.990479
8	20	24	0.849167	0.986772

In [27]:

```
params = {
 'max_depth' : [8, 16, 20],
 'min_samples_split' : [8, 16, 24]
}
grid_cv = GridSearchCV(dt_HAR, param_grid = params, scoring = 'accuracy', cv = 5, return_train_score = True)
grid_cv.fit(X_train, Y_train)
```

In [28]:

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

In [29]:

```
print('최고 평균 정확도: {0:.4f}, 최적 하이퍼 매개변수: {1}'.format(grid_cv.best_score_, grid_cv.best_params_))
```

In [30]:

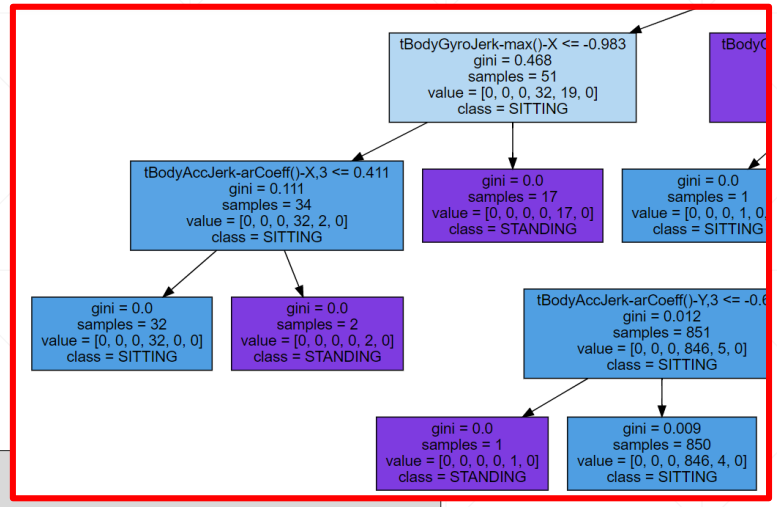
```
best_dt_HAR = grid_cv.best_estimator_
best_Y_predict = best_dt_HAR.predict(X_test)
best_accuracy = accuracy_score(Y_test, best_Y_predict)
print('best 결정 트리 예측 정확도: {0:.4f}'.format(best_accuracy))
```

결과 시각화

8. 결정 트리 모델을 이용한 각 피처의 중요도 확인

In [31]:	import seaborn as sns import matplotlib.pyplot as plt
In [32]:	feature_importance_values = best_dt_HAR.feature_importances_ feature_importance_values_s = pd.Series(feature_importance_values, index = X_train.columns)
In [33]:	feature_top10 = feature_importance_values_s.sort_values(ascending = False)[:10]
In [34]:	plt.figure(figsize = (10, 5)) plt.title('Feature Top 10') sns.barplot(x = feature_top10, y = feature_top10.index) plt.show()

결과 시각화 (cont'd)



9. 결정 트리 모델의 트리 구조를 그림으로 시각화하기

In [35]:

```
!pip install graphviz
```

In [36]:

```
# https://www2.graphviz.org/Packages/stable/windows/10/cmake/Release/x64/
##graphviz-install-2.44.1-win64.exe를 다운로드하여 설치하기

import os
###설치 경로(C:/Program Files/Graphviz 2.44.1/bin)를 PATH에 추가하기
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz 2.44.1/bin'
```

In [37]:

```
from sklearn.tree import export_graphviz
#export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일 생성
export_graphviz(best_dt_HAR, out_file = "tree.dot", class_names = label_name,
feature_names = feature_name, impurity = True, filled = True)
```

In [38]:

```
import graphviz
#위에서 생성된 tree.dot 파일을 Graphviz가 읽어서 시각화
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```