11. 사이키런_분류 분석

박경미

목차

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 분석 미리보기

특징 데이터로 유방암 진단하기	
목표	로지스틱 회귀 분석을 이용해 유방암에 영향을 미치는 특징 데이터를 분석하고 유방암 여부를 진단하는 예측 모델을 생성한다.
핵심 개념	로지스틱 회귀, 시그모이드 함수, 성능 평가 지표, 오차 행렬, 정밀도, 재현율, F1 스코어, ROC 기반 AUC 스코어
데이터 준비	유방암 진단 데이터: 사이킷런 내장 데이터셋
데이터 탐색	1. 사이킷런 데이터셋에서 제공하는 설명 확인: b_cancer.DESCR 2. 사이킷런 데이터셋에 지정된 X 피처와 타깃 피처 결합 3. 로지스틱 회귀 분석을 위해 X 피처 값을 정규 분포 형태로 스케일링: b_cancer_scaled = scaler.fit_transform(b_cancer.data)
분석 모델 구축	사이킷런의 로지스틱 회귀 모델 구축
결과 분석	성능 평가 지표 계산: confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 목표설정

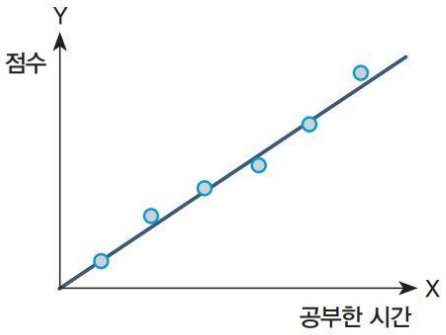
- **목표: 유방암 특징을 측정한 데이터에 로지스틱 회귀 분석을 수행하여 유방암 발생을 예측**

❖ 핵심 개념 이해

- **로지스틱 회귀**
  - 분류에 사용하는 기법으로 선형 회귀와 달리 S자 함수를 사용하여 참(True, 1)과 거짓(False, 0)을 분류
- **시그모이드 함수**
  - 로지스틱 회귀에서 사용하는 S자 함수
  - x의 값이 커지면 y의 값은 1에 근사하게 되고 x의 값이 작아지면 y의 값은 0에 근사하게 되어 S자 형태의 그래프가 됨
  - 두 개의 값을 분류하는 이진 분류에 많이 사용

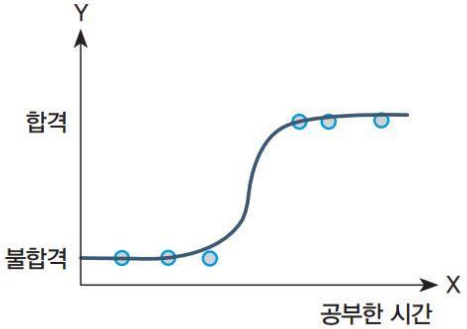
• 방정식 $$y = \frac{1}{1+e^{ax+b}}$$

공부 시간	1	3	5	7	9	11
점수	40	55	65	70	80	95



(a) 선형 회귀와 선형 함수

공부 시간	1	3	5	7	9	11
점수	불합격	불합격	불합격	합격	합격	합격



(b) 로지스틱 회귀와 S자 함수

그림 11-1 선형 회귀와 로지스틱 회귀 비교

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 핵심 개념 이해

▪ 로지스틱 회귀

- 선형 회귀 모델은 실제값과 예측값의 오차에 기반한 지표를 사용
- 로지스틱 회귀 모델은 이진 분류 결과를 평가하기 위해 오차 행렬에 기반한 성능 지표인 정밀도, 재현율, F1 스코어,
- ROC_AUC를 사용

▪ 오차 행렬

- 행렬을 사용해 이진 분류의 예측 오류를 나타내는 지표
- 행은 **실제 클래스**의 Negative/Positive 값 / 열은 **예측 클래스의** Negative/ Positive

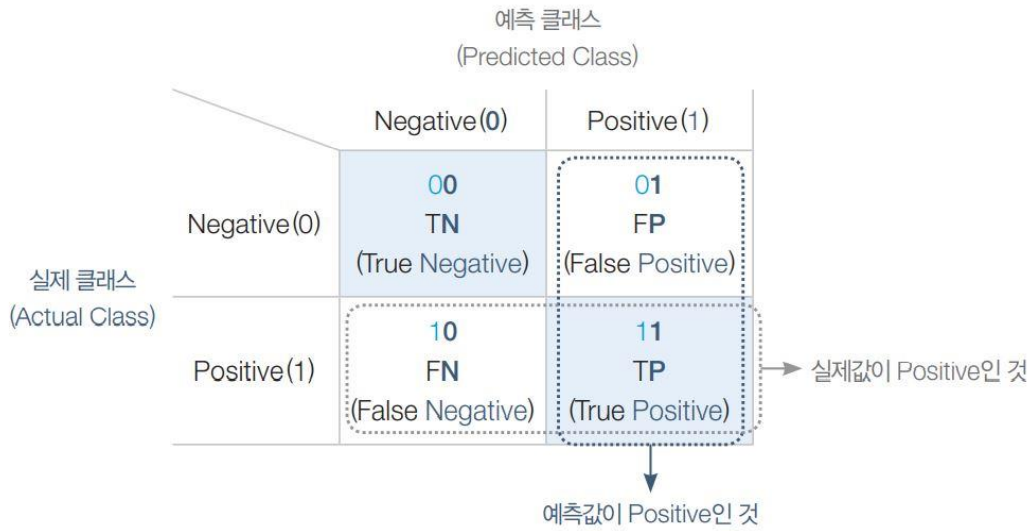


그림 11-2 오차 행렬

TN: Negative가 참인 경우 TP: Positive가 참인 경우

FN: Negative가 거짓인 경우 FP: Positive가 거짓인 경우

사이킷런에서는 오차 행렬을 구하기 위해 confusion_matrix 함수를 제공

- 정확도 $= \frac{\text{예측 결과와 실제값이 동일한 건수}}{\text{전체 데이터 수}} = \frac{(TN+TP)}{(TN+FP+FN+TP)}$

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 핵심 개념 이해

▪ 정밀도

- 예측이 Positive인 것(FP+TP) 중에서, 참인 것(TP)의 비율을 의미
- 정밀도는 Positive 예측 성능을 더 정밀하게 평가하기 위한 지표로 사용
- 사이킷런에서는 정밀도를 구하기 위해 precision_score 함수를 제공

- 정밀도 $= \frac{TP}{(FP+TP)}$

▪ 재현율

- 실제값이 Positive인 것(FN+TP) 중에서 참인 것(TP)의 비율을 의미
- 실제 Positive인 데이터를 정확히 예측했는지 평가하는 지표 (민감도 또는 TPR)
- 사이킷런에서는 재현율을 구하기 위해 recall_score 함수를 제공

- 재현율 $= \frac{TP}{(FN+TP)}$

▪ F1 스코어

- 정밀도와 재현율을 결합한 평가 지표
- 정밀도와 재현율이 서로 트레이드 오프 관계(상충 관계)인 문제점을 고려하여 정확한 평가를 위해 많이 사용
- 사이킷런에서는 F1 스코어를 구하기 위해 f1_score 함수를 제공

- F1 스코어 $= \frac{2}{\frac{1}{\text{재현율}}+\frac{1}{\text{정밀도}}} = 2\times\frac{\text{정밀도}\times\text{재현율}}{\text{정밀도}+\text{재현율}}$

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 핵심 개념 이해

▪ ROC 기반 AUC 스코어

- 오차 행렬의 FPR이 변할 때 TPR이 어떻게 변하는지를 나타내는 곡선
  - FPR: 실제 Negative인 데이터를 Positive로 거짓False으로 예측한 비율
  - TPR: 실제 Positive인 데이터를 참True으로 예측한 비율(재현율)

- $\text{FPR} = \frac{\text{FP}}{(\text{FP}+\text{TN})}$

- ROC 기반의 AUC 값은 ROC 곡선 밑의 면적을 구한 것으로 1에 가까울수록 좋은 성능을 의미
- 사이킷런에서는 ROC 기반의 AUC를 구하기 위해 roc_auc_score 함수를 제공

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 데이터 준비 및 탐색

▪ 사이킷런에서 제공하는 데이터셋

표 11-1 사이킷런에서 제공하는 주요 데이터셋

데이터셋	샘플 갯수	독립 변수	종속 변수	데이터 로드 함수
보스톤 주택 가격 데이터	506	13개	주택 가격	load_boston()
붓꽃(아이리스) 데이터	150	4개	붓꽃 종류: setosa, versicolor, virginica	load_iris()
당뇨병 환자 데이터	442	10개	당뇨병 수치	load_diabetes()
숫자 0~9를 손으로 쓴 흑백 데이터	1797	64개	숫자: 0~9	load_digits()
와인의 화학 성분 데이터	178	13개	와인 종류: 0, 1, 2	load_wine()
체력 검사 데이터	20	3개	체력 검사 점수	load_linnerud()
유방암 진단 데이터	569	30개	악성(malignant), 양성(benign): 1, 0	load_ breast_ cancer()

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 데이터 준비 및 탐색

▪ 사이킷런의 유방암 진단 데이터셋 사용하기

1. 데이터 준비하기

In [1]:	**import** numpy **as** np **import** pandas **as** pd **from** sklearn.datasets **import load_breast_cancer**
In [2]:	b_cancer = **load_breast_cancer()**

In [1]: 사이킷런에서 제공하는 데이터셋sklearn.datasets중에서 유방암 진단 데이터셋을 사용하기 위해 load_breast_cancer를 임포트
In [2]: 데이터셋을 로드하여 객체b_cancer를 생성

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 데이터 준비 및 탐색

▪ 사이킷런의 유방암 진단 데이터셋 사용하기

2. 데이터 탐색하기

In [3]:
```
print(b_cancer.DESCR)
```

In [4]:
```
b_cancer_df = pd.DataFrame(b_cancer.data, columns = b_cancer.feature_names)
```

In [5]:
```
b_cancer_df['diagnosis']= b_cancer.target
```

In [6]:
```
b_cancer_df.head()
```

Out:[6]

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374

5 rows × 31 columns

In [3]: 데이터셋에 대한 설명을 확인
In [4]: 데이터셋 객체의 data 배열b_cancer.data, 즉 독립 변수 X가 되는 피처를 DataFrame 자료형으로 변환하여 b_cancer_df를 생성
In [5]: 유방암 유무 class로 사용할 diagnosis 컬럼을 b_cancer_df에 추가하고 데이터셋 객체의 target 컬럼b_cancer.target을 저장
In [6]: b_cancer_df의 데이터 샘플 5개를 출력b_cancer_df.head( )하여 확인

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 데이터 준비 및 탐색

▪ 사이킷런의 유방암 진단 데이터셋 사용하기

3. 데이터셋의 크기와 독립 변수 X가 되는 피처에 대한 정보를 확인

In [7]:
```
print('유방암 진단 데이터셋 크기: ', b_cancer_df.shape)
```

Out:[7]
```
유방암 진단 데이터셋 크기: (569, 31)
```

In [8]:
```
b_cancer_df.head()
```

Out:[8]
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
mean radius 569 non-null float64
mean texture 569 non-null float64
mean perimeter 569 non-null float64
mean area 569 non-null float64
mean smoothness 569 non-null float64
mean compactness 569 non-null float64
mean concavity 569 non-null float64
mean concave points 569 non-null float64
mean symmetry 569 non-null float64
mean fractal dimension 569 non-null float64
radius error 569 non-null float64
texture error 569 non-null float64
....
```

Out:[8]
```
....
perimeter error 569 non-null float64
area error 569 non-null float64
smoothness error 569 non-null float64
compactness error 569 non-null float64
concavity error 569 non-null float64
concave points error 569 non-null float64
symmetry error 569 non-null float64
fractal dimension error 569 non-null float64
worst radius 569 non-null float64
worst texture 569 non-null float64
worst perimeter 569 non-null float64
worst area 569 non-null float64
worst smoothness 569 non-null float64
worst compactness 569 non-null float64
worst concavity 569 non-null float64
worst concave points 569 non-null float64
worst symmetry 569 non-null float64
worst fractal dimension 569 non-null float64
diagnosis 569 non-null int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

In [7]: b_cancer_df.shape를 사용하여 데이터셋의 행의 개수(데이터 샘플 개수)와 열의 개수(변수 개수)를 확인
행의 개수가 569이므로 데이터 샘플이 569개, 열의 개수가 31이므로 변수가 31개 있음

In [8]: b_cancer_df에 대한 정보를 확인 b_cancer_df.info( ) / 30개의 피처(독립 변수 X) 이름과 1개의 종속 변수 이름을 확인 가능
diagnosis는 악성이면 1, 양성이면 0의 값이므로 유방암 여부에 대한 이진 분류의 class로 사용할 종속 변수가 됨

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 데이터 준비 및 탐색

▪ 사이킷런의 유방암 진단 데이터셋 사용하기

4. 로지스틱 회귀 분석에 피처로 사용할 데이터를 평균이 0, 분산이 1이 되는 정규 분포 형태로 맞춤

In [9]:	from sklearn.preprocessing import StandardScaler scaler = StandardScaler()
In [10]:	b_cancer_scaled = scaler.fit_transform(b_cancer.data)
In [11]:	print(b_cancer.data[0])
Out:[11]	[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01 4.601e-01 1.189e-01]
In [12]:	print(b_cancer_scaled[0])
Out:[12]	[1.09706398 -2.07333501 1.26993369 0.9843749 1.56846633 3.28351467 2.65287398 2.53247522 2.21751501 2.25574689 2.48973393 -0.56526506 2.83303087 2.48757756 -0.21400165 1.31686157 0.72402616 0.66081994 1.14875667 0.90708308 1.88668963 -1.35929347 2.30360062 2.00123749 1.30768627 2.61666502 2.10952635 2.29607613 2.75062224 1.93701461]

In [9]: 사이킷런의 전처리 패키지에 있는 정규 분포 스케일러를 임포트하고 사용할 객체scaler를 생성
In [10]: 피처로 사용할 데이터b_cancer.data에 대해 정규 분포 스케일링을 수행scaler.fit_ transform( )하여 b_cancer_scaled에 저장
In [11]~[12]: 정규 분포 스케일링 후에 값이 조정된 것을 확인

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 분석 모델 구축 및 결과 분석

1. 로지스틱 회귀를 이용하여 분석 모델 구축하기

In [13]:	**from** sklearn.linear_model import **LogisticRegression** **from** sklearn.model_selection **import** train_test_split
In [14]:	*#X, Y 설정하기* Y = b_cancer_df['diagnosis'] X = b_cancer_scaled
In [15]:	*#훈련용 데이터와 평가용 데이터 분할하기* X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 0)
In [16]:	*#로지스틱 회귀 분석: (1) 모델 생성* lr_b_cancer = **LogisticRegression()**
In [17]:	*#로지스틱 회귀 분석: (2) 모델 훈련* lr_b_cancer.fit(X_train, Y_train)
Out:[17]	LogisticRegression()
In [18]:	*#로지스틱 회귀 분석: (3) 평가 데이터에 대한 예측 수행 -> 예측 결과 Y_predict 구하기* Y_predict = lr_b_cancer.**predict**(X_test)

In [13]: 필요한 모듈을 임포트
In [14]: diagnosis를 Y, 정규 분포로 스케일링한 b_cancer_scaled를 X로 설정
In [15]: 전체 데이터 샘플 569개를 학습 데이터:평가 데이터=7:3으로 분할test_size=0.3함
In [16]: 로지스틱 회귀 분석 모델 객체lr_b_cancer를 생성
In [17]: 학습 데이터X_train, Y_train로 모델 학습을 수행fit( )함
In [18]: 학습이 끝난 모델에 대해 평가 데이터 XX_test를 가지고 예측을 수행predict( ) 하여 예측값 YY_predict를 구함

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 분석 모델 구축 및 결과 분석

▪ 로지스틱 회귀를 이용하여 분석 모델 구축하기

- 주피터 노트북 버전 확인
  - 실습하는 아나콘다의 주피터 노트북 버전에 따라 실행 결과가 조금 다르게 나타날 수 있음
  - 노트북 화면 상단의 [Help]- [About] 메뉴에서 확인

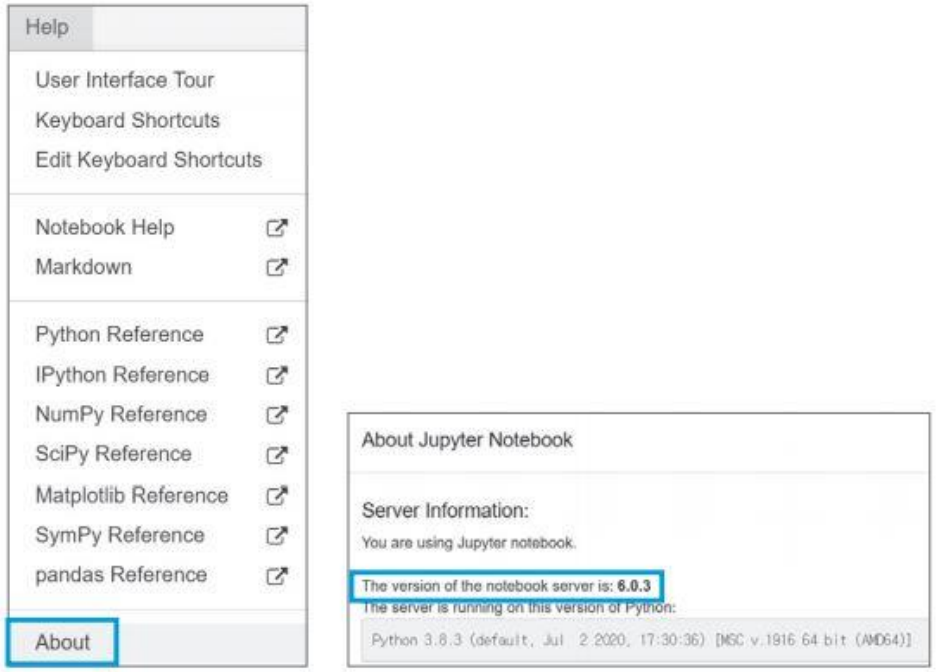


그림 11-3 주피터 노트북 버전 확인

01. [로지스틱 회귀 분석] 특징 데이터로 유방암 진단하기

❖ 분석 모델 구축 및 결과 분석

2. 생성한 모델의 성능 확인하기

In [19]:	from sklearn.metrics import confusion_matrix, accuracy_score from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
In [20]:	*#오차 행렬* confusion_matrix(Y_test, Y_predict)
Out:[20]	array([[60, 3], [1, 107]], dtype = int64)
In [21]:	acccuracy = accuracy_score(Y_test, Y_predict) precision = precision_score(Y_test, Y_predict) recall = recall_score(Y_test, Y_predict) f1 = f1_score(Y_test, Y_predict) roc_auc = roc_auc_score(Y_test, Y_predict)
In [22]:	print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f}, F1: {3:.3f}'.format(acccuracy,precision,recall,f1))
Out:[22]	정확도: 0.977, 정밀도: 0.973, 재현율: 0.991, F1: 0.982
In [23]:	print('ROC_AUC: {0:.3f}'.format(roc_auc))
Out:[23]	ROC_AUC: 0.972

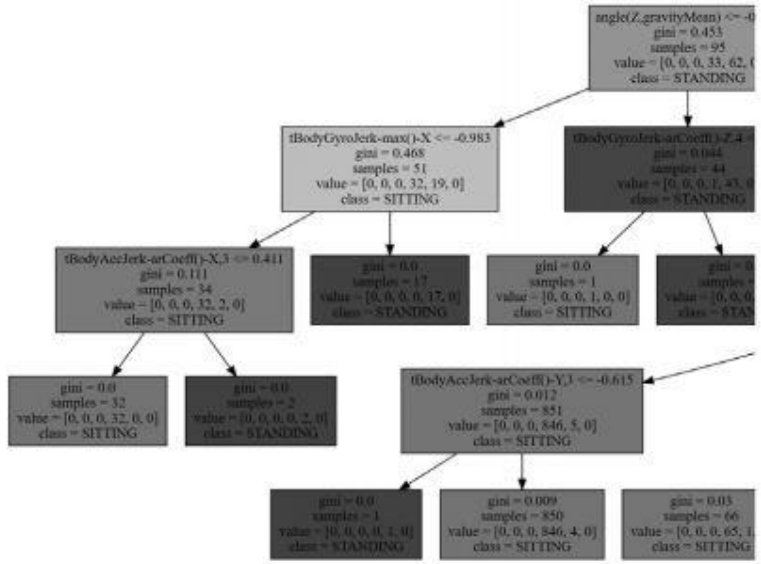
In [19]: 필요한 모듈을 임포트
In [20]: 평가를 위해 7:3으로 분할한 171개의 test 데이터에 대해 이진 분류의 성능 평가 기본이 되는 오차 행렬을 구함. 실행 결과를 보면 TN이 60개, FP가 3개, FN이 1개, TP가 107개인 오차 행렬이 구해 짐
In [21]: 성능 평가 지표인 정확도, 정밀도, 재현율, F1 스코어, ROC-AUC 스코어를 구함
In [22]~[23]: 성능 평가 지표를 출력하여 확인

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 미리보기

센서 데이터로 움직임 분류하기	
목표	스마트폰으로 수집한 센서 데이터를 분석하여 사람의 움직임에 대한 분류 모델을 생성하고 새로운 데이터에 대한 움직임 유형을 예측하여 분류한다.
핵심 개념	결정 트리, 정보 이득 지수, 지니 계수, Graphviz 패키지
데이터 준비	센서 데이터: UCI Machine Learning Repository에서 다운로드
데이터 탐색	1. 피처 이름 파일을 로드하여 객체로 저장 2. 훈련 데이터셋을 파일에서 로드하여 객체로 저장 3. 평가 데이터셋을 파일에서 로드하여 객체로 저장 4. 레이블 이름 파일을 로드하여 객체로 저장
분석 모델 구축	사이킷런의 결정 트리 모델 구축: 결정 트리 모델의 생성, 훈련, 예측
결과 분석	1. 성능 평가 지표 계산: accuracy_score 2. 결정 트리의 하이퍼 매개변수 변경에 대한 정확도 분석 3. 최적 결정 트리 모델 생성: GridSearchCV 4. 중요 피처 분석: feature_importances_

결과 시각화
Graphviz 패키지를 사용한 트리 시각화



02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 목표 설정

- **목표: 스마트폰에서 수집한 센서 데이터를 분석하여 사람의 움직임을 분류 하는 모델을 생성 새로운 데이터에 대해 움직임 유형을 예측해서 분류**

❖ 핵심 개념 이해

- **결정 트리**
  - 머신러닝 알고리즘 중에서 직관적으로 이해하기 쉬운 것으로 다중 분류에 많이 사용
  - 데이터 안에서 if/else 기반으로 규칙을 찾아 학습하여 트리 구조의 분류 규칙을 만듬
  - 결정 트리의 구조는 규칙 조건(if)을 나타내는 규칙 노드와 분류가 결정된 클래스 값이 표시된 리프 노드로 구성
  - 데이터의 균일도를 계산하는 대표 적인 방법으로 정보 이득 지수와 지니 계수가 있음

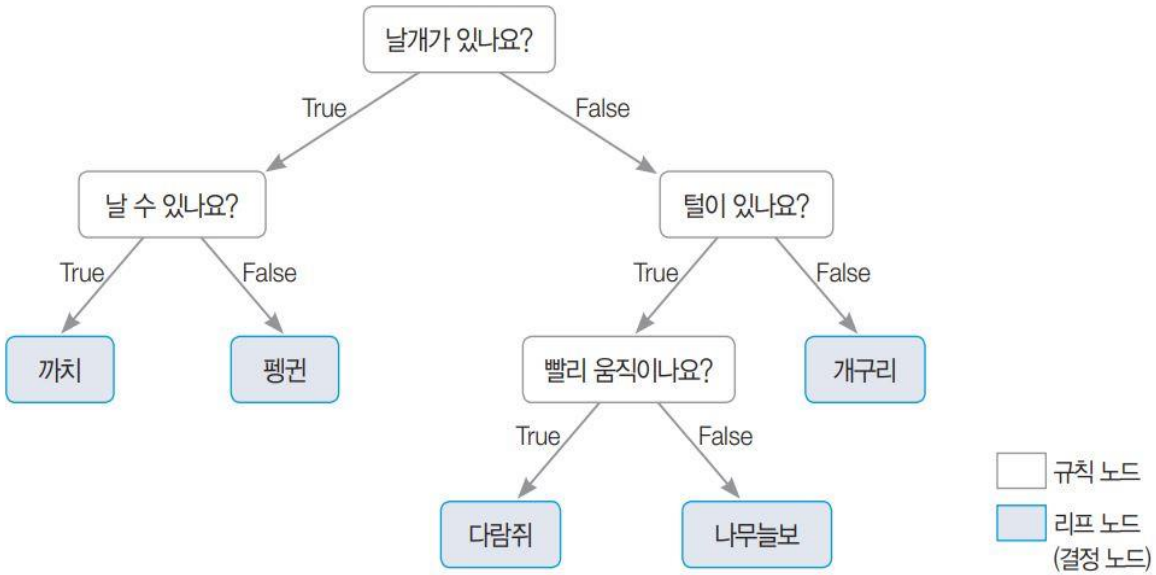


그림 11-4 {개구리, 펭귄, 까치, 나무늘보, 다람쥐}를 분류하기 위한 결정 트리

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 핵심 개념 이해

▪ 정보 이득 지수

- 정보 이득은 엔트로피 개념을 기반으로 함
  - 엔트로피: 데이터 집합의 혼잡도를 의미
  - 데이터 집합에 다른 데이터 = 균일도가 떨어짐 → 혼잡도가 높아지므로 엔트로피가 높아짐
  - 데이터 집합에 같은 데이터 = 균일도가 높아짐 → 혼잡도가 떨어지므로 엔트로피가 낮아짐
- 정보 이득 지수: 혼잡도가 줄어들어 얻게 되는 이득을 의미하는 것으로, '1-엔트로피'로 계산
- 결정 트리: 정보 이득 지수가 높은 피처를 분할 기준으로 사용

▪ 지니 계수

- 소득의 불균형 정도를 나타내는 것인데 머신러닝에서 지니 계수는 데이터의 순도를 나타내기 위해 사용
- 결정 트리에서는 지니 계수가 높을수록 순도가 낮은 데이터 집합을 의미
- 지니 계수가 0이면 완전 순수한 데이터 집합을 의미

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 핵심 개념 이해

▪ DecisionTreeClassifie

- 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: $\log_2$(전체 피처 개수)를 계산하여 설정
max_depth	트리의 최대 깊이
max_leaf_nodes	리프 노드에 들어가는 샘플 데이터의 최대 개수

▪ Graphviz

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

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 준비

1. 센서 데이터 다운로드하기

1. UCI Machine Learning Repository에 접속하여 'human activity recognition'을 검색

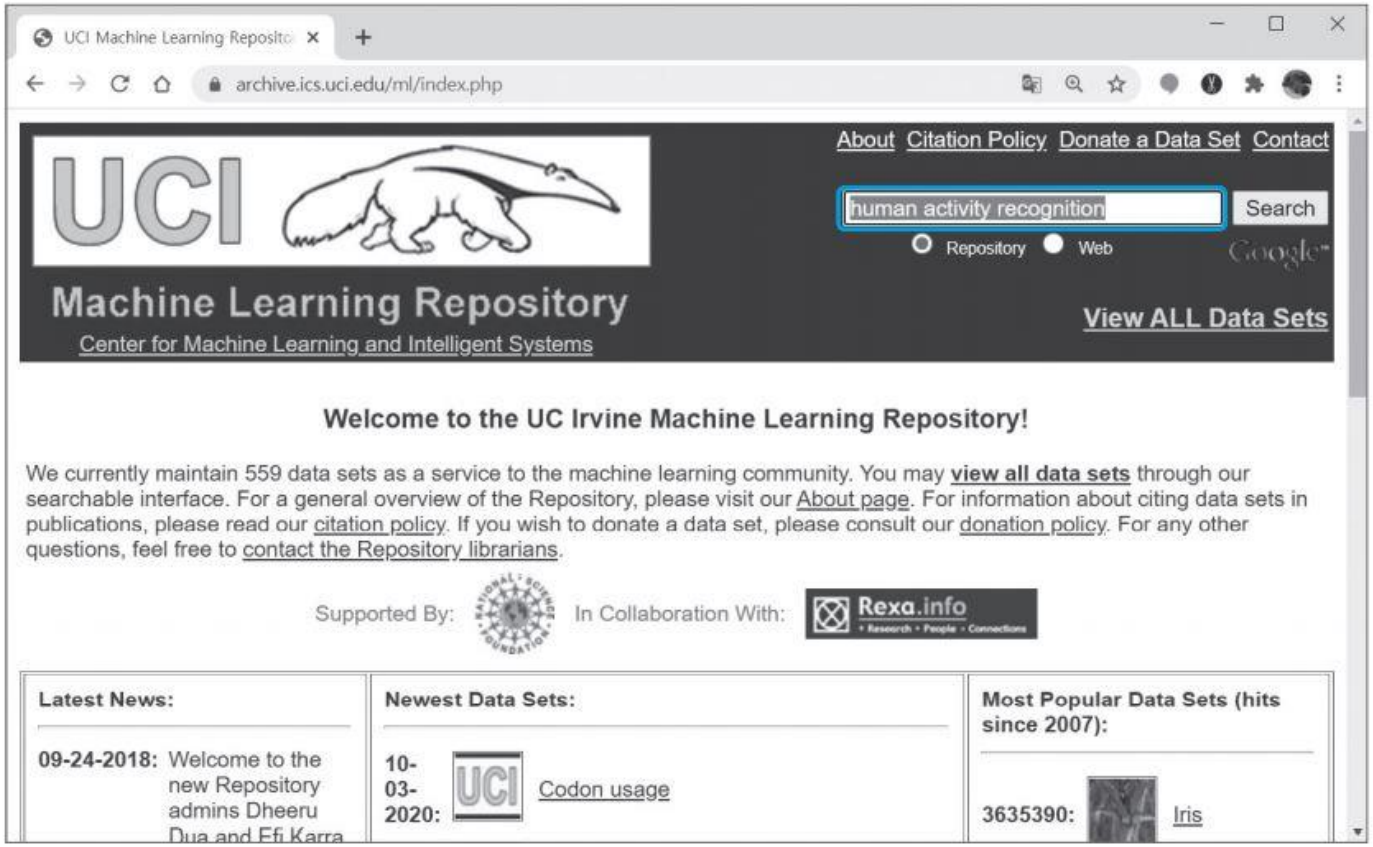


그림 11-5 UCI Machine Learning Repository 사이트에서 'human activity recognition' 검색

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 준비

1. 센서 데이터 다운로드하기

2. 검색 결과 목록에서 'Human Activity Recognition Using Smartphones Data Set'을 클릭

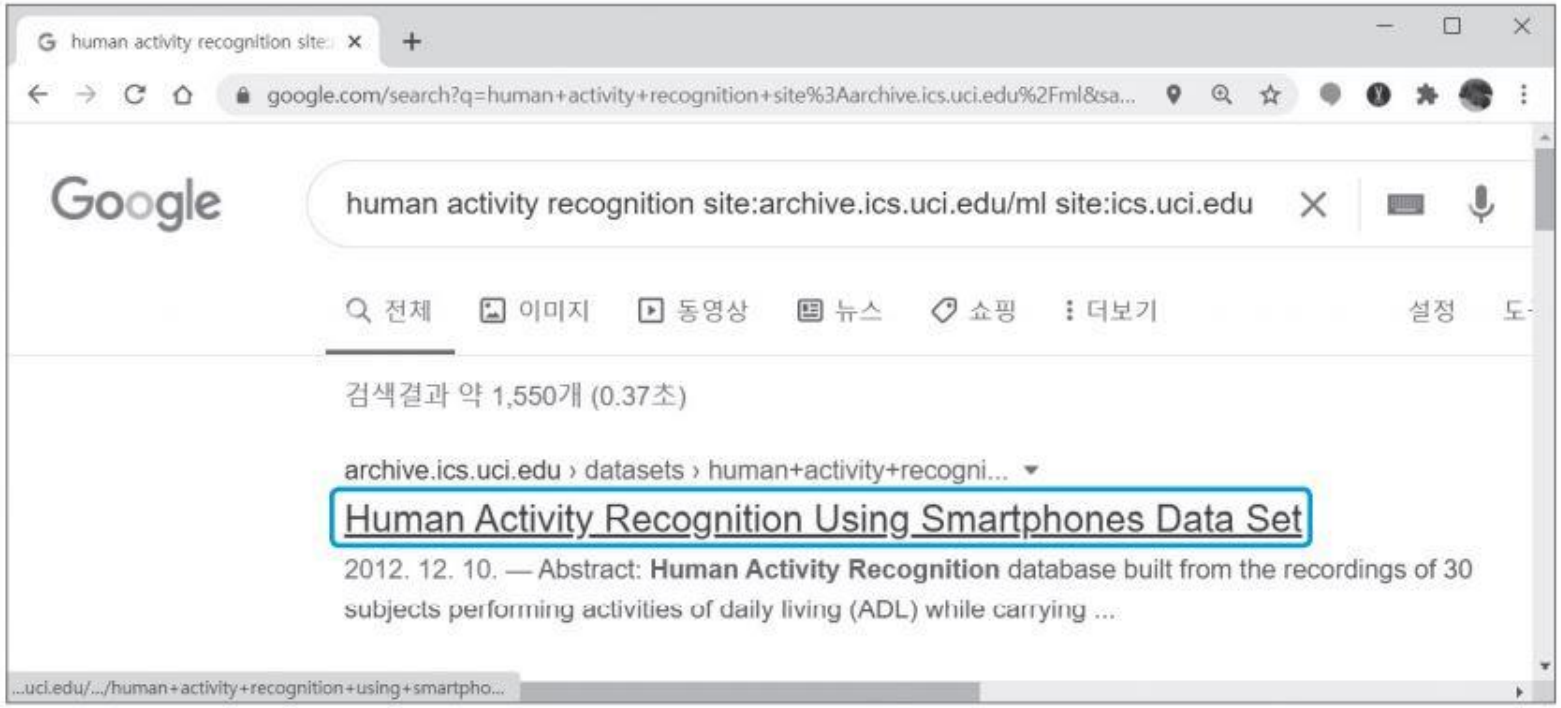


그림 11-6 검색 목록에서 다운로드할 데이터셋 선택

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 준비

1. 센서 데이터 다운로드하기

3. 'Human Activity Recognition Using Smartphones Data Set' 페이지가 나타나면 'Data Folder'를 클릭하고, 나타난 페이지에서 'UCI HAR Dataset.zip'을 클릭하여 다운로드

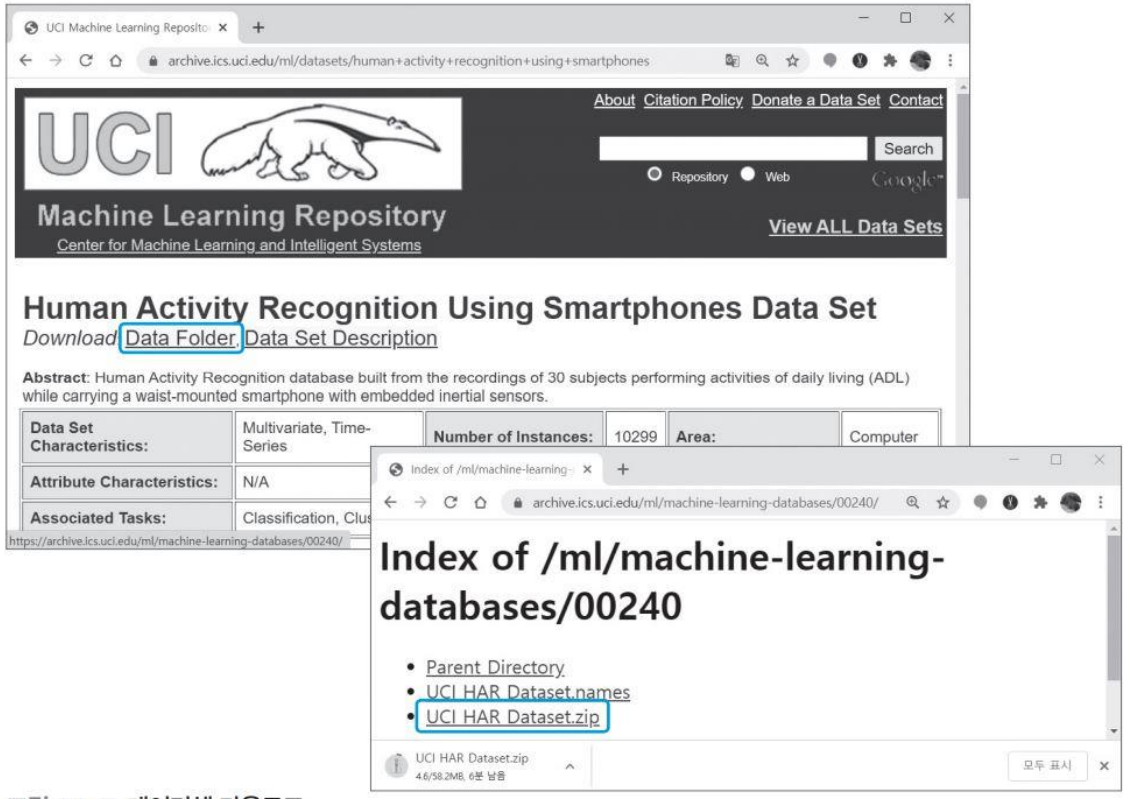


그림 11-7 데이터셋 다운로드

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 준비

1. 센서 데이터 다운로드하기

4. 'My_Python 폴더에 11장_data 폴더를 만든 뒤 다운로드한 'UCI HAR Dataset.zip' 파일을 옮기고 폴더 이름을 'UCI_HAR_Dataset'으로 변경(하위 폴더의 이름도 함께 변경)

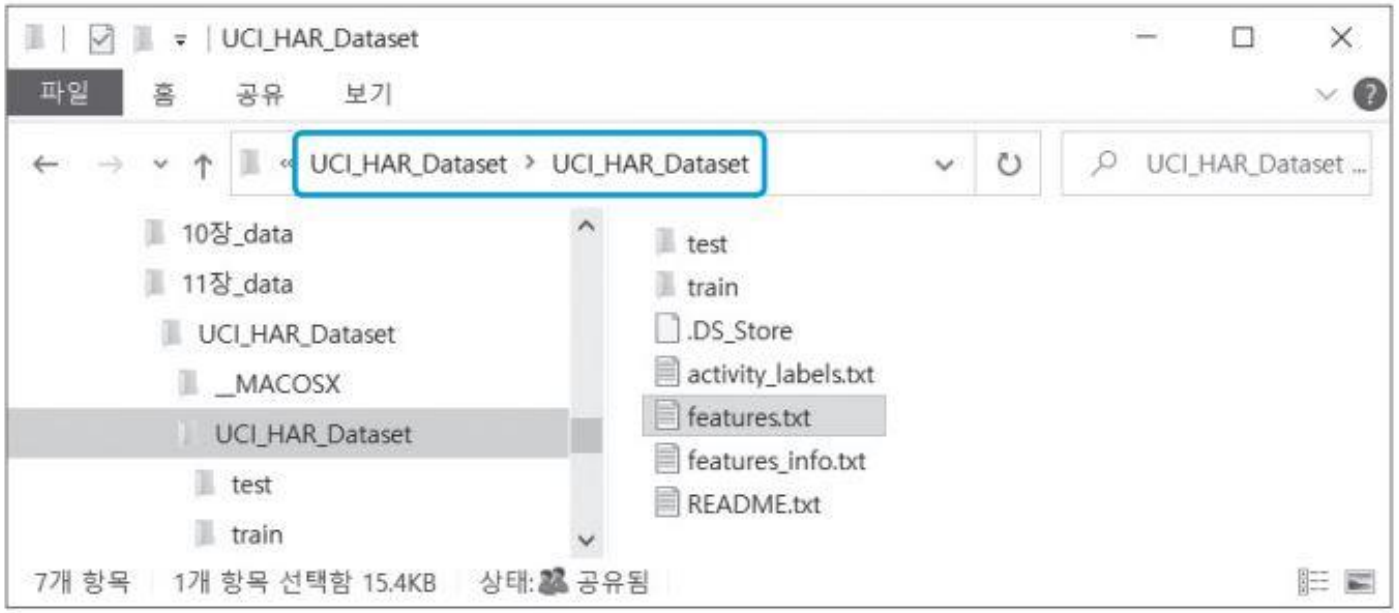


그림 11-8 다운로드한 데이터셋의 이름 변경

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 탐색

▪ 훈련용과 테스트용 데이터셋 확인하기

- 다운로드한 데이터에 대한 설명은 README.txt 파일에 나와있음
- activity_labels.txt를 보면 WALKING, WALKING_UPSTAIRS, WALKING_ DOWNSTAIRS, SITTING, STANDING, LAYING과 같은 6가지 움직임이 있는 것을 확인
- 결정 트리 모델을 사용해서 이 6가지 움직임을 분류

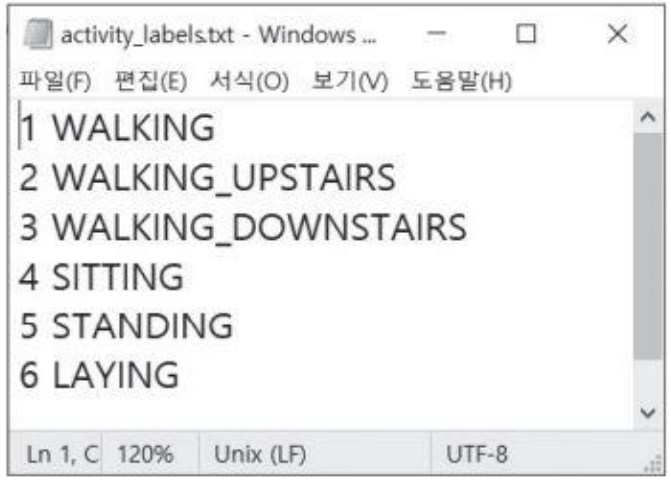


그림 11-9 다운로드한 파일 확인

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 탐색

▪ 훈련용과 테스트용 데이터셋 확인하기

1. 주피터 노트북에서 '결정트리분석'으로 노트북 페이지를 추가하고 입력

In [1]:	**import** numpy **as** np **import** pandas **as** pd pd.__version__
Out:[1]	'0.24.2'

In [1]: 필요한 모듈을 임포트하고 설치되어 있는 pandas 버전을 확인

- 오류체크 (ValueError)
  - 확인한 pandas 버전이 0.24.2보다 높은 경우 'ValueError : Duplicate names are not allowed.'라는 오류가 발생
  - features_name에 중복된 이름이 있는데 pandas 최신 버전에서는 이를 금지하고 있어서 발생
  - 다음 명령어를 입력하여 컬럼 이름 중복을 허용하는 하위 버전으로 pandas를 다시 설치

```
!pip install pandas = = 0.24.2
```

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 탐색

▪ 훈련용과 테스트용 데이터셋 확인하기

1. 주피터 노트북에서 '결정트리분석'으로 노트북 페이지를 추가하고 입력

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

In [3]:
```
feature_name_df.head()
```

Out:[3]

	index	feature_name
0	1	tBodyAcc-mean()-X
1	2	tBodyAcc-mean()-Y
2	3	tBodyAcc-mean()-Z
3	4	tBodyAcc-std()-X
4	5	tBodyAcc-std()-Y

In [4]:
```
feature_name_df.shape
```

Out:[4]
```
(561, 2)
```

In [5]:
```
#index 제거하고, feature_name만 리스트로 저장
feature_name = feature_name_df.iloc[:, 1].values.tolist()
```

In [6]:
```
feature_name[:5]
```

Out:[6]
```
['tBodyAcc-mean()-X',
 'tBodyAcc-mean()-Y',
 'tBodyAcc-mean()-Z',
 'tBodyAcc-std()-X',
 'tBodyAcc-std()-Y']
```

In [2]~[5]: 피처 이름이 있는 features.txt 파일을 열어서 내용을 확인, 561개 피처가 있는데 feature_name만 추출해서 리스트로 저장

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 탐색

▪ 훈련용과 테스트용 데이터셋 확인하기

2. X_train, X_test, Y_train, Y_test 데이터 파일도 분석에 사용할 수 있도록 준비

In [7]:	X_train = pd.read_csv('data/UCI_HAR_Dataset/UCI_HAR_Dataset/train/X_train.txt', sep='₩s+', names = feature_name, engine = 'python') X_test = pd.read_csv('data/UCI_HAR_Dataset/UCI_HAR_Dataset/ test/X_test.txt', sep='₩s+', names = feature_name, engine = 'python') Y_train = pd.read_csv('data/UCI_HAR_Dataset/UCI_HAR_Dataset/train/y_train.txt', sep='₩s+', header = None, names = ['action'], engine = 'python') Y_test = pd.read_csv('data/UCI_HAR_Dataset/UCI_HAR_Dataset/test/y_test.txt' , sep = '₩s+', header = None, names = ['action'], engine = 'python')
Out:[7]	C:₩Users₩kmj₩Anaconda3₩lib₩site-packages₩pandas₩io₩parsers.py:702: UserWarning: Duplicate names specified. This will raise an error in the future. return _read(filepath_or_buffer, kwds)
In [8]:	X_train.shape, Y_train.shape, X_test.shape, Y_test.shape
Out:[8]	((7352, 561), (7352, 1), (2947, 561), (2947, 1)

In [7]~[8]: train 폴더와 test 폴더에는 훈련용 X/Y 데이터와 테스트용 X/Y 데이터가 txt 파일로 들어 있음
파일을 읽어서 저장하고 크기를 확인하면X_train.shape, Y_train.shape, X_test. shape, Y_test.shape 훈련용 데이터는 7,352개,
테스트용 데이터는 2,947개로 구성된 것을 확인 가능

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 탐색

▪ 훈련용과 테스트용 데이터셋 확인하기

2. X_train, X_test, Y_train, Y_test 데이터 파일도 분석에 사용할 수 있도록 준비

In [9]:
```
X_train.info()
```

Out:[9]
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7352 entries, 0 to 7351
Columns: 561 entries, tBodyAcc-mean()-X to angle(Z,gravityMean)
dtypes: float64(561)
memory usage: 31.5 MB
```

In [10]:
```
X_train.head()
```

Out:[10]

	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-meanFreq()	fBodyl
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.074323	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.158075	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	0.414503	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	0.404573	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.087753	

5 rows × 561 columns

In [11]:
```
print(Y_train['action'].value_counts())
```

Out:[11]
```
6 1407
5 1374
4 1286
1 1226
2 1073
3 986
Name: action, dtype: int64
```

In [9]~[10]: 훈련용 X 데이터는 feature_name에서 확인했던 561개 피처로 구성되어 있음

In [11]: Y 데이터는 6가지 움직임에 대한 레이블(분류할 class)값으로 되어있으므로, 각 레이블의 데이터 개수value_counts( )를 확인

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 데이터 탐색

▪ 훈련용과 테스트용 데이터셋 확인하기

2. X_train, X_test, Y_train, Y_test 데이터 파일도 분석에 사용할 수 있도록 준비

In [12]:	label_name_df = pd.read_csv('data/UCI_HAR_Dataset/UCI_HAR_Dataset/activity_labels.txt', sep = '₩s+', header = None, names = ['index', 'label'], engine = 'python')

In [13]:	#index 제거하고, feature_name만 리스트로 저장
	label_name = label_name_df.iloc[:, 1].values.tolist()

In [14]:	label_name
Out:[14]	['WALKING', 'WALKING_UPSTAIRS', 'WALKING_DOWNSTAIRS', 'SITTING', 'STANDING', 'LAYING']

In [12]: 레이블 이름이 있는 파일인 activity_labels.txt에서 label_name만 추출해 리스트로 저장

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 모델 구축 및 결과 분석

1. 결정 트리 분류 분석 모델 구축하기

- 6개 움직임을 분류하기 위한 결정 트리 모델을 구축

In [15]:	**from** sklearn.tree import **DecisionTreeClassifier**
In [16]:	dt_HAR = DecisionTreeClassifier(random_state=156) *#결정 트리 분류 분석: 모델 생성*
In [17]:	dt_HAR.**fit**(X_train, Y_train) *#결정 트리 분류 분석: 모델 훈련*
Out:[17]	DecisionTreeClassifier(class_weight = None, criterion = 'gini', max_depth = None,max_features = None, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_weight_fraction_leaf = 0.0, presort = False, random_state = 156, splitter = 'best')

In [18]:	#결정 트리 분류 분석: 평가 데이터에 예측 수행 -> 예측 결과로 Y_predict 구하기
	Y_predict = dt_HAR.predict(X_test)

In [15]: 사이킷런을 사용하여 결정 트리 분류 분석을 하기 위해 sklearn.tree 패키지에 있는 DecisionTreeClassifier 모듈을 임포트
In [16]: 훈련용 데이터와 테스트용 데이터는 이미 준비되어 있으므로 모델 생성 작업을 수행
In [17]: 모델 훈련을 수행, 훈련이 끝나고 출력된 결정 트리 모델의 매개변수에서 riterion = 'gini'는 분할 기준으로 지니 계수를 사용한다는 의미
In [18]: 평가 데이터로 예측을 수행하고 예측값을 Y_predict에 저장

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 모델 구축 및 결과 분석

2. 결정 트리 분류 분석 모델 구축하기

1. 결과 분석하기 - 생성된 결정 트리 모델의 분류 정확도 성능을 확인

In [19]:	**from** sklearn.metrics **import** accuracy_score
In [20]:	accuracy = accuracy_score(Y_test, Y_predict) print('결정 트리 예측 정확도: {0:.4f}'.format(accuracy))
Out:[20]	결정 트리 예측 정확도: 0.8548
In [21]:	print('결정 트리의 현재 하이퍼 매개변수: ₩n', dt_HAR.get_params())
Out:[21]	결정 트리의 현재 하이퍼 매개변수: {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'presort': 'deprecated', 'random_state': 156, 'splitter': 'best'}

In [19]: 정확도 측정을 위해 accuracy_score 모듈을 임포트
In [20]: 테스트용 데이터의 Y_test 값과 결정 트리 모델에서 예측한 Y_predict의 오차를 기반으로 계산한 정확도 점수를 확인
In [21]: 결정 트리 모델 학습을 통해 자동 설정되어 있는 하이퍼 매개변수를 확인

- 결정 트리 모델의 하이퍼 매개변수를 수정하면 정확도를 높일 수 있음

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 모델 구축 및 결과 분석

2. 결정 트리 분류 분석 모델 구축하기

2. 정확도를 검사하여 최적의 하이퍼 매개변수를 찾는 작업을 해주는 GridSearchCV 모듈을 사용

In [22]:	**from** sklearn.model_selection **import GridSearchCV**

In [23]:	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)

Out:[23]	GridSearchCV(cv = 5, estimator = DecisionTreeClassifier(random_state= 156), param_grid = {'max_depth': [6, 8, 10, 12, 16, 20, 24]}, return_train_score = True, scoring = 'accuracy')

In [22]: GridSearchCV 모듈을 임포트

In [23]: GridSearchCV를 사용하여 결정 트리의 하이퍼 매개변수 중에서 트리의 깊이를 6, 8, 10, 12, 16, 20, 24로 변경하면서 결정 트리 모델 7개를 생성하여 모델 학습grid_cv.fit( )을 수행

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 모델 구축 및 결과 분석

2. 결정 트리 분류 분석 모델 구축하기

2. 정확도를 검사하여 최적의 하이퍼 매개변수를 찾는 작업을 해주는 GridSearchCV 모듈을 사용

In [24]:
```
cv_results_df = pd.DataFrame(grid_cv.cv_results_)
cv_results_df[['param_max_depth', 'mean_test_score', mean_train_score']]
```

Out:[24]

	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	0.851311	0.999660
5	20	0.850800	0.999966
6	24	0.849440	1.000000

In [25]:
```
print('최고 평균 정확도: {0:.4f}, 최적 하이퍼 매개변수: {1}'.format(grid_cv.best_score_, grid_cv.best_params_))
```

Out:[25]
```
최고 평균 정확도: 0.8513, 최적 하이퍼 매개변수: {'max_depth': 16}
```

In [24]: GridSearchCV를 사용하여 생성한 7개 모델의 param_max_depth, mean_test_ score, mean_train_score를 확인

In [25]: 7개 모델 중에서 최고 평균 정확도와 그때의 최적 max_depth를 출력하여 확인

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 모델 구축 및 결과 분석

2. 결정 트리 분류 분석 모델 구축하기

3. max_depth와 함께 min_samples_split을 조정

In [26]:	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)

Out:[26]	GridSearchCV(cv = 5, estimator = DecisionTreeClassifier(random_state = 156), param_grid = {'max_depth': [8, 16, 20], 'min_samples_split': [8, 16, 24]}, return_train_score = True, scoring = 'accuracy')

In [26]: max_depth를 8, 16, 20으로, min_samples_split를 8, 16, 24로 변경하면서 결정 트리 모델을 생성하여 모델 학습grid_cv.fit( )을 수행

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 모델 구축 및 결과 분석

2. 결정 트리 분류 분석 모델 구축하기

3. max_depth와 함께 min_samples_split을 조정

In [27]:

```
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']]
```

Out:[27]

	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 [28]:

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

Out:[28]

```
최고 평균 정확도: 0.8549, 최적 하이퍼 매개변수: {'max_depth': 8, 'min_samples_split': 16}
```

In [27]: GridSearchCV를 사용하여 생성한 9개 모델의 param_max_depth, min_ samples_split, mean_test_score, mean_train_score를 확인

In [28]: GridSearchCV를 사용하여 생성한 모델 중에서 최고 평균 정확도와 최적 하이퍼 매개변수를 출력하여 확인

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 모델 구축 및 결과 분석

2. 결정 트리 분류 분석 모델 구축하기

4. 최적 모델grid_cv.best_estimator_을 사용하여 테스트 데이터에 대한 예측을 수행

In [27]:	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))
Out:[27]	best 결정 트리 예측 정확도: 0.8717

In [29]: GridSearchCV의 객체인 grid_cv의 best_estimator_ 속성에 저장되어 있는 최적 모델best_dt_HAR에 대하여 테스트 데이터X_test에 대한 예측predict( )을 수행하고 정확도를 출력하여 확인

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 모델 구축 및 결과 분석

2. 결정 트리 분류 분석 모델 구축하기

5. feature_importances_ 속성을 사용하여 각 피처의 중요도를 알아내고 중요도가 높은 10개 피처를 찾아 그래프로 표시

In [30]:	import seaborn as sns import matplotlib.pyplot as plt
In [31]:	feature_importance_values = best_dt_HAR.feature_importances_ feature_importance_values_s = pd.Series(feature_importance_values, index = X_train.columns)
In [32]:	feature_top10 = feature_importance_values_s.sort_values(ascending = False)[:10]
In [33]:	plt.figure(figsize = (10, 5)) plt.title('Feature Top 10') sns.barplot(x = feature_top10, y = feature_top10.index) plt.show()

In [30]: 중요 피처를 그래프로 나타내기 위한 모듈을 임포트

In [31]: 최적 결정 트리 모델best_dt_HAR의 feature_importances_를 객체에 저장하고 막대 그래프로 그리기 위해 Series 자료형으로 변환하여 저장

In [32]: 중요도 값을 오름차순 정렬하여 상위 10개만 feature_top10에 저장

In [33]: 중요 피처 10개를 막대 그래프로 나타냄

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 분석 모델 구축 및 결과 분석

2. 결정 트리 분류 분석 모델 구축하기

5. feature_importances_ 속성을 사용하여 각 피처의 중요도를 알아내고 중요도가 높은 10개 피처를 찾아 그래프로 표시

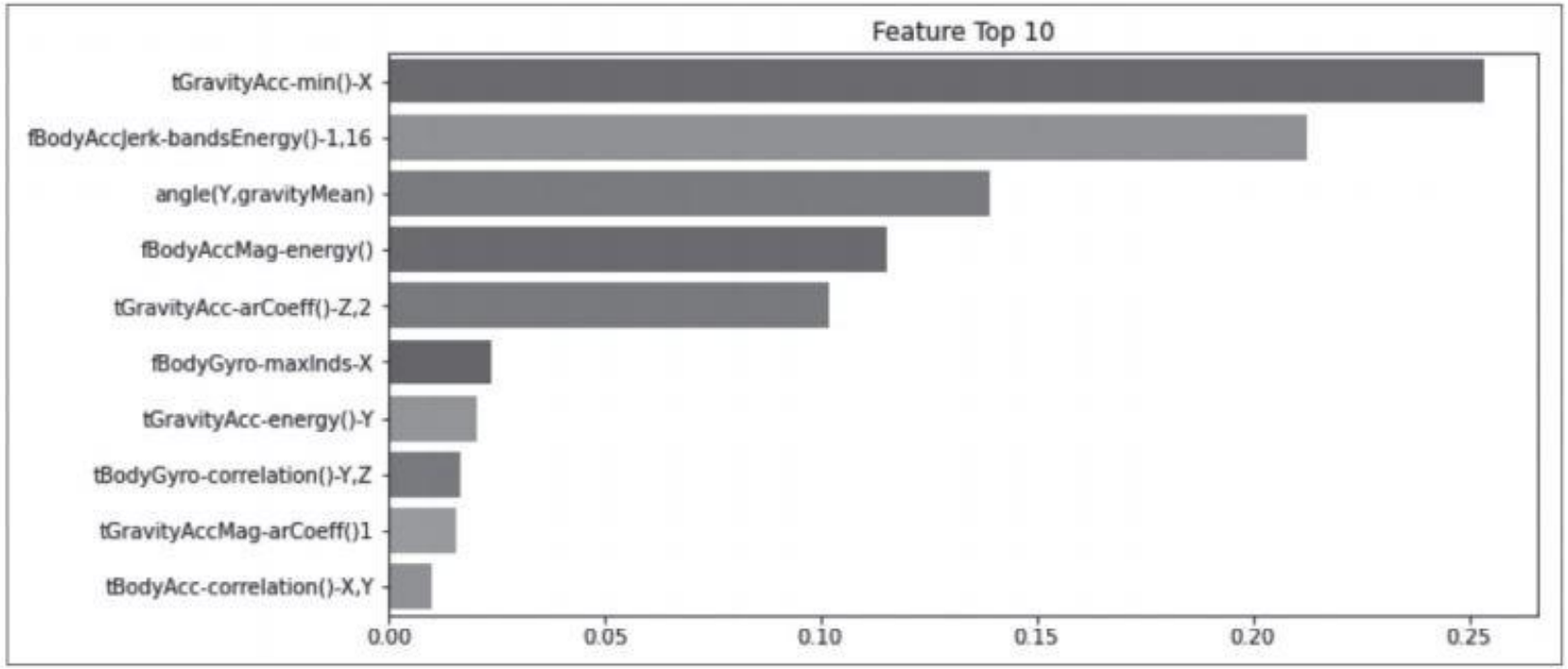


그림 11-10 중요도 값 상위 10개 피처로 나타낸 막대 그래프

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 결과 시각화

1. 결정 트리 모델의 트리 구조를 그림으로 시각화하기

1. Graphviz 패키지는 별도의 설치 작업이 필요
   사이트에서 graphviz-install-2.44.1-win64.exe를 다운로드하여 설치

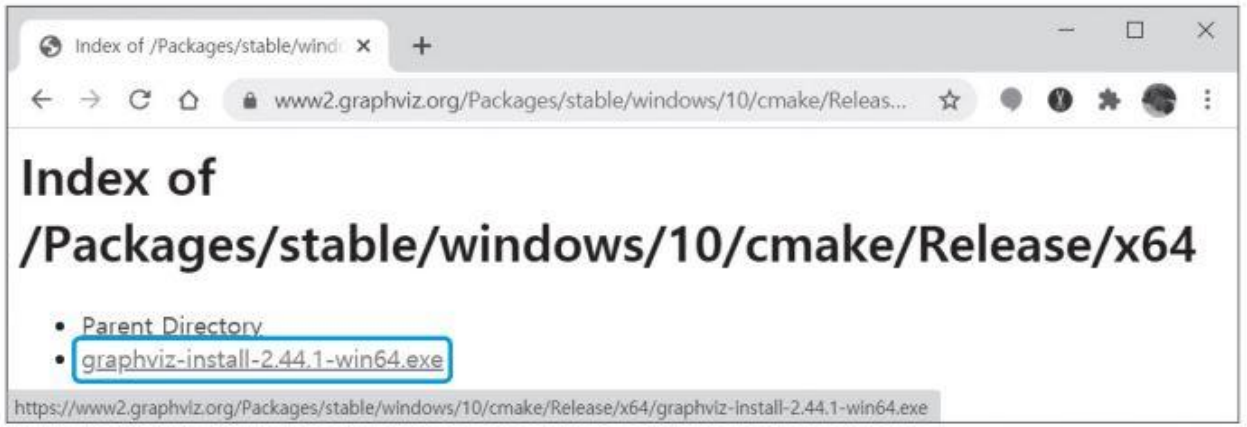


그림 11-11 Graphviz 패키지 다운로드 후 설치

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 결과 시각화

1. 결정 트리 모델의 트리 구조를 그림으로 시각화하기

2. 설치 경로를 환경 변수에 설정하고 Graphviz 패키지를 파이썬으로 사용하기 위해 파이썬 래퍼 모듈인 graphviz를 설치

In [34]:	*#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 [35]:	!pip install **graphviz**
In [36]:	**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 [37]:	import graphviz with open("tree.dot") as f: *#위에서 생성된 tree.dot 파일을 Graphviz가 읽어서 시각화* dot_graph = f.read() graphviz.Source(dot_graph)

In [34]: Graphviz 2.44.1을 설치한 경로를 환경 변수로 설정
In [35]: Graphviz를 사용하기 위해 파이썬 래퍼 모듈을 설치
In [36]: Graphviz 인터페이스 모듈인 export_graphviz를 임포트, 결정 트리 모델best_dt_HAR의 트리 구조 정보를 dot 파일로 생성
In [37]: dot 파일을 읽어서 트리 구조를 그림으로 나타냄

02. [결정 트리 분석 + 산점도/선형 회귀 그래프] 센서 데이터로 움직임 분류하기

❖ 결과 시각화

1. 결정 트리 모델의 트리 구조를 그림으로 시각화하기

- 시각화된 트리 그래프를 보면 561개 피처를 사용하여 depth가 8인 결정 트리를 작성한 것을 확인

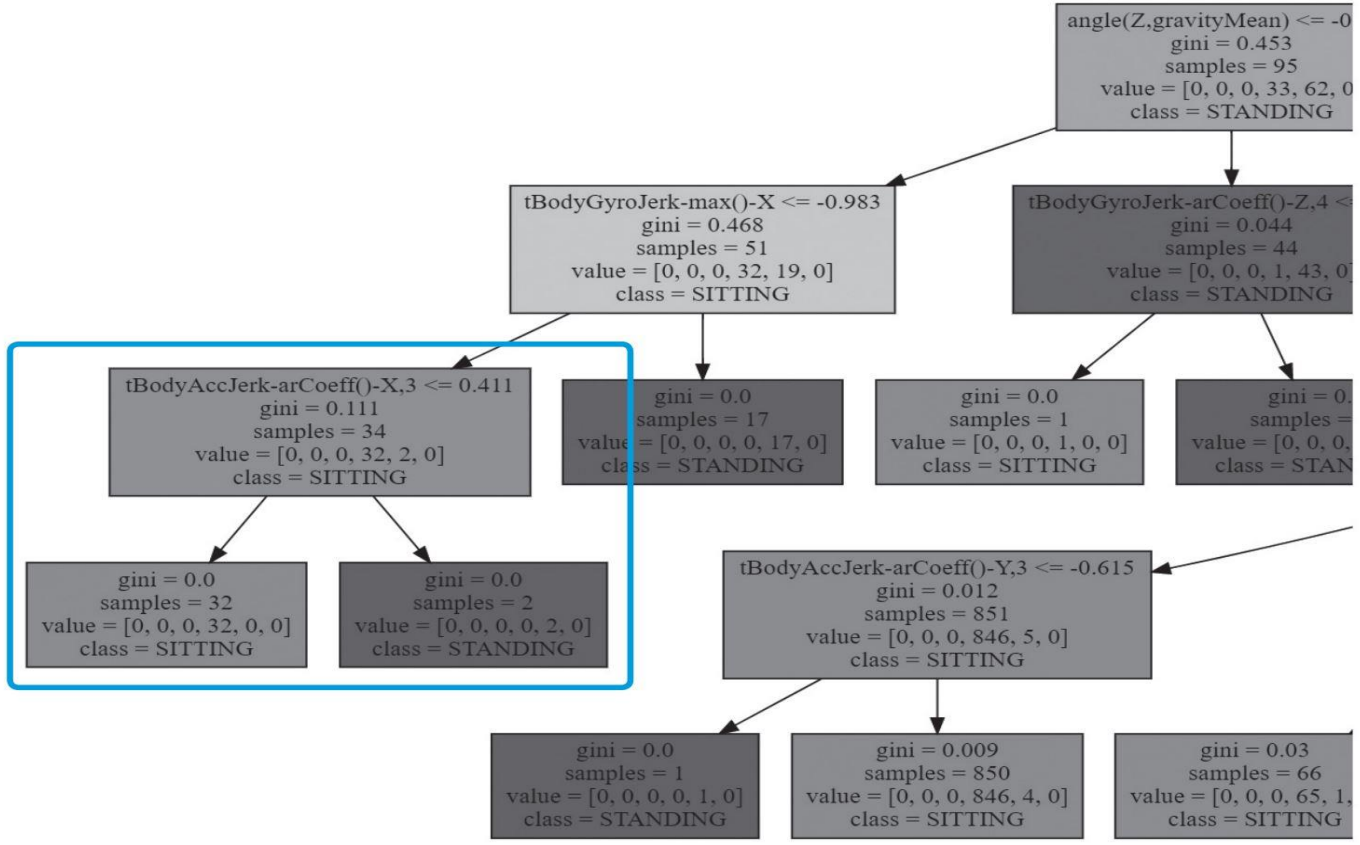


그림 11-12 결정 트리 모델을 시각화한 트리 구조 그래프