3주차_4장. 분류 - Part 1(4.1 ~ 4.4장)

상태	진행 중
마감일	@02/28/2025

4. 분류(Classification)
    4.1 분류의 개요
    4.2 결정 트리
        결정 트리 모델의 특징
        결정 트리 파라미터
        결정 트리 모델의 시각화
        결정 트리 과적합(Overfitting)
        결정 트리 실습 - 사용자 행동 인식 데이터 세트
    4.3 앙상블 학습
        앙상블 학습 개요
        보팅 유형 - 하드 보팅 / 소프트 보팅
        보팅 분류기(Voting Classifier)
    4.4 랜덤 포레스트
        랜덤 포레스트 개요 및 실습
        랜덤 포레스트 하이퍼 파라미터, 튜닝

4. 분류(Classification)

4.1 분류의 개요

- 지도학습의 대표적 유형
- 피처&레이블값 → 학습, 모델 생성 → 새 데이터 값의 레이블 예측

▼ 다양한 Classification 머신러닝 알고리즘

- 나이브 베이즈 - 베이트 통계 기반
- Logistic Regression - 변수들 간 선형 관계성 기반
- Decision Tree - 데이터 균일도에 따른 규칙 기반

- Support Vector Machine - 개별 클래스 간 최대 분류 마진 탐색
- Nearest Neighbor(NN) - 근접 거리 기준
- Neural Network(심층 연결 기반 신경망)
- Ensemble - 여러 알고리즘 결합

앙상블 - Bagging

- Random Forest
  - 예측 성능 ↑, 수행 시간 ↓, 유연성

앙상블 - Boosting

- 근래 Boosting으로 발전 중!
- Gradient Boosing
  - 예측 성능 ↑, 수행 시간 ↑

    → 최적화 모델 튜닝 어려움
- XGBoost, LIghtBGM 등으로 해결!

앙상블 - Stacking

- 앙상블의 앙상블

- 서로 다른/ 같은 알고리즘 결합하지만. 보통 동일 알고리즘 결합 - Decision Tree
  - 장점 : 데이터 스케일링, 정규화 등 사전 가공의 영향 적음
  - 단점 : 성능 향상을 위한 복잡한 규칙 구조로 overfitting 위험

    → 이 점은 앙상블에서 오히려 장점이 됨

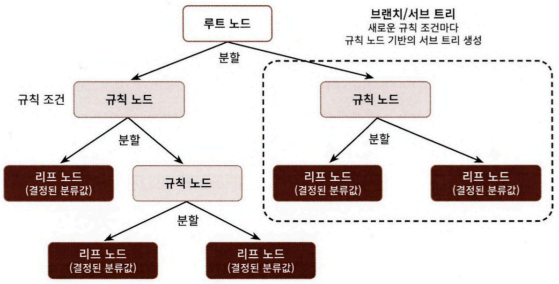
    : 앙상블은 여러 약한 학습기(성능 ↓)결합해 가중치를 업데이트하는데, 이때 DT가 좋은 약한 학습기

4.2 결정 트리

먼저 그래서 결정 트리에 대해 알아보자

> 💡 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리 기반의 분류 규칙을 만드는 것

- 보통 if/else 기반



- Root node : 처음 시작 노드
- Decision Node(규칙 노드) : 규칙 조건이 됨
  - 새로운 규칙 조건마다 = 데이터의 피처가 결합해 규칙을 만들 때마다 Sub Tree 생성
- Branch Node : 자식 노드 O 노드
- Leaf Node : 최종 노드. 결정된 클래스 값

※ 많은 규칙 = 복잡 = depth(깊이) 깊을수록 → 성능 저하

→ 가능한 적은 결정 노드

→ 데이터 분류 시 최대한 많은 데이터가 해당 분류에 속할 수 있게 규칙 지정

→ " 어떻게 트리를 Split분할할 것인가? "

: 균일도가 **높은 것** 먼저 선택하는 조건 생성

- 균일도와 규칙 조건
  - 혼잡도 ↔ 균일도
- 정보 이득 지수(Information Gain)
  - 엔트로피 : 데이터의 혼잡도
  - ⇒ **(1-엔트로피 지수)**
  - **높은 속성**을 기준으로 분할
- 지니 계수
  - 불평등 지수
  - 0 = 평등, 1 = 불평등
  - **낮은 속성**을 기준으로 분할

  +사이킷런 DecisionTreeClassifier가 이용
- Information Gain이나 지니 계수로 조건 찾아 자식 트리 노드 반복 분할,

  데이터가 모두 특정 분류에 속하면 stop (recursive한 방법)

결정 트리 모델의 특징

- (+) 균일도만 신경 쓰면 돼 특수한 경우 제외 **피처 스케일링 / 정규화 등 전처리 필요 X**
- (-) 계속된 노드 분할로 과적합 방지 위해 트리 **크기 사전 제한**

결정 트리 파라미터

DecisionTreeClassifier, DecisionTreeRegressor 클래스 제공,

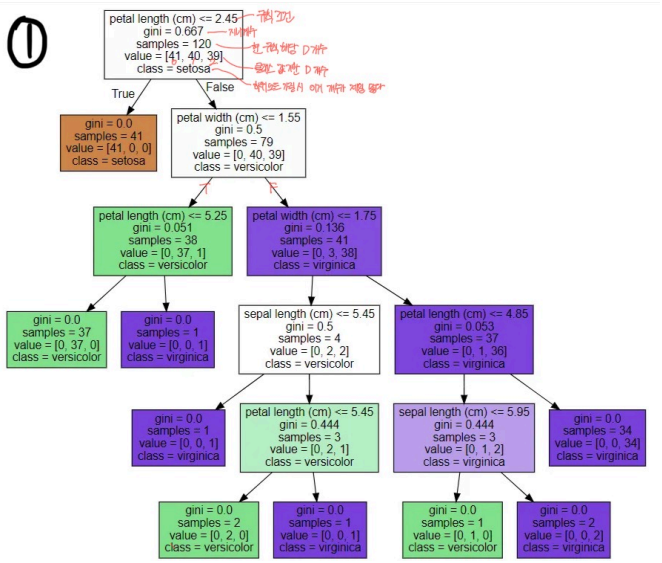
CART(classification and regression tress) 알고리즘 기반(분류+회귀 둘 다 可)

1. `min_samples_split`
   - 노드를 분할하기 위한 최소한의 데이터 수
   - Default = 2 / 작을수록 분할되는 노드 ↑, 과적합 가능성 ↑ (과적합 제어에 사용)
2. `min_samples_leaf`
   - 분할 시 양쪽 노드가 가져야 할 최소한의 데이터 수
   - 클수록 만족시키기 어려움 → 노드 분할 ↓ (과적합 제어에 사용)

   +) 주의 : 비대칭적 데이터일 경우 특정 클래스 데이터 극도로 작을 수 있어 이 경우 작게 설정
3. `max_features`
   - 분할 시 고려할 최대 피처 개수
   - Default = None : 모든 피처 사용해 분할. 전체 피처 선정
     - int형 지정 : 대상 피처의 **개수**
     - float형 지정 : 전체 피처 중 대상 피처의 **퍼센트**
     - 'sqrt', 'auto' : 전체 피처 중 **√(전체 피처 개수)**
     - 'log' : 전체 피처 중 log2(전체 피처 개수)
4. `max_depth`
   - 트리의 최대 깊이
   - Default = None. (완벽하게 될 때까지 하거나 `min_samples_split` 보다 작아질때까지 하거나)
   - 과적합 주의
5. `max_leaf_nodes`
   - leaf node의 최대 개수

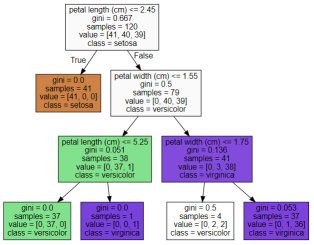
결정 트리 모델의 시각화

- Graphviz 패키지 사용
- 사이킷런의 `export_graphviz()` API와 인터페이스



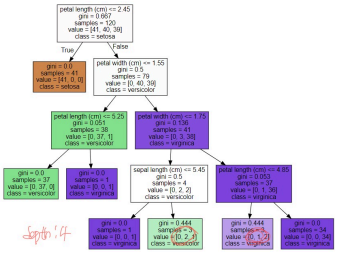
루트 노드에 기술된 지표의 의미 필기참고

- max depth 0 → 3 제한 시



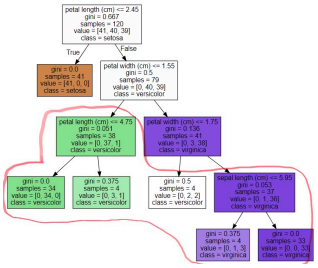
깊이 3까지만 생성

- min_samples_leaf 0→ 4



동그라미 친 부분 : 데이터 3개 이므로 split 안함

- min_samples_split 0 → 4



분할 시 자식 노드 2개 다 4 이상이어야 하므로 더이상 X인 노드 발생

- `feature_importance_` 속성 - 피처의 역할 지표
  - ndarray 형태로 값 반환해 피처 순서대로 값 할당
  - 정보 이득 / 지니 계수를 얼마나 잘 개선시켰는지 정규화된 값으로 표현
  - 값 ↑, 중요도 ↑

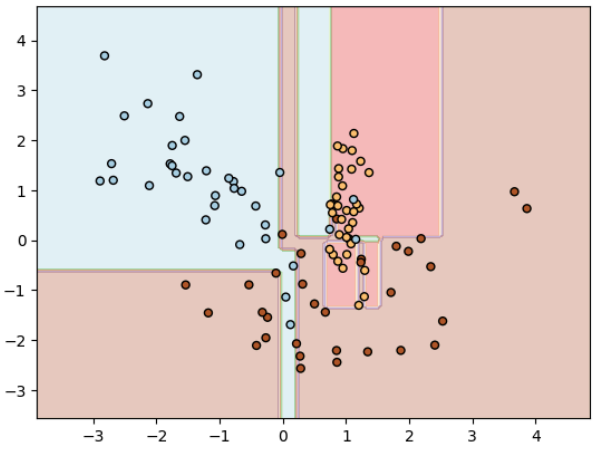
결정 트리 과적합(Overfitting)

- `make_classification()` 함수 : 분류 위한 테스트용 데이터 생성
  - 호출 시 반환되는 객체 = 피처 데이터 세트, 클래스 레이블 데이터 세트
  - 트리 생성에 별다른 제약 X

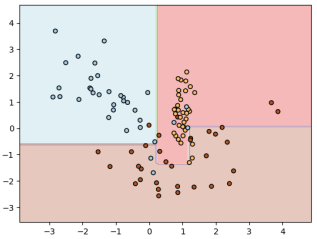
```
X_features, y_labels = make_classification(
    n_features=2,        # 피처 개수 → 2차원 시각화
    n_redundant=0,       # 불필요하게 상관관계 높은 피처(중복 정보)는 없음
    n_informative=2,     # 실제 분류에 유용한 정보가 되는 피처 수 → 2개
    n_classes=3,         # 클래스 개수 → 3종류
    n_clusters_per_class=1, # 각 클래스가 하나의 클러스터만 형성
    random_state=0       # 난수 고정
)
```

```
plt.scatter(
    X_features[:,0],     # X축 좌표 (첫 번째 피처)
    X_features[:,1],     # Y축 좌표 (두 번째 피처)
    marker='o',          # 점 모양 동그라미
    c=y_labels,          # 점 색깔을 클래스 값(y_labels)에 따라 다르게
    s=25,                # 점 크기
    edgecolor='k'        # 점 테두리 색 검정(black)
)
```

- `visualize_boundary()` 함수
  - 트리가 어떠한 결정 기준을 가지고 분할하는지 확인
  - 모델이 클래스 값을 예측하는 기준을 색상&경계로 표시



outlier 포함 확인



#min_samples_leaf = 6

이게 더 성능 뛰어날 것
- train에만 지나치게 최적화되지 X

결정 트리 실습 - 사용자 행동 인식 데이터 세트

UCI 머신러닝 리포지토리 제공 '사용자 행동 데이터 세트(Human Activity Recognition' 예측 분류 해보기

- 스마트폰 센서 부착 후 사람 동작 관련 여러 feature 수집한 데이터

```
#feature.txt 파일에는 피처 이름 index와 피처명이 공백으로 분리되어 있음. 이를 df로 로드
feature_name_df = pd.read_csv('features.txt', sep='\s+',header=None, names=['column_index', 'column_name'])

#피처명 index 제거하고, 피처명만 리스트 객체로 생성한 뒤 샘플로 10개만 추출
feature_name = feature_name_df.iloc[:,1].values.tolist()
print('전체 피처명에서 10개만 추출 : ',feature_name[:10])
```

💡 전체 피처명에서 10개만 추출 : ['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']

인체 움직임 관련 속성 평균/표준편차가 X,Y,Z축 값으로 돼 있음

1. 중복된 피처명 확인
2. 원본 피처명에 _1 또는 _2를 추가로 부여
   - `get_new_feature_name_df()` 함수
3. 학습 D 세트, 레이블 D 세트, 테스트용 피처 데이터 파일, 레이블 데이터 파일 Df 로드
4. train D 살펴보기
   - 7352개 레코드, 561개 피처, 모두 float형(인코딩 필요x)
   - 레이블 값 1,2,3,4,5,6, 분포도 고름
5. 예측 분류 수행
   - 85.48% 정확도

   a. depth가 정확도에 주는 영향

- `max_depth` = [6,8,10,12,14,16,20,24]로 GridSeachCV 실행
- 결과 : 85.48%, `max_depth` = 8, `min_samples_split` = 16
- 각 depth의 정확도 확인

	param_max_depth	mean_test_score
0	6	0.847662
1	8	0.854879
2	10	0.852705
3	12	0.845768
4	14	0.848351
5	16	0.847127
6	20	0.848624
7	24	0.848624

max_depth 8 넘어가며 정확도 계속 떨어짐. 깊어질수록 overfittin 可 알 수 O

b. 별도의 test 세트에서 정확도 측정

- 별도 세트에서 `min_samples_split` = 16 고정, `max_depth` 변화에 따른 값 측정
- 각 depth 정확도 확인

```
max_depth = 6 정확도 : 0.8551
max_depth = 8 정확도 : 0.8717
max_depth = 10 정확도 : 0.8599
max_depth = 12 정확도 : 0.8571
max_depth = 16 정확도 : 0.8599
max_depth = 20 정확도 : 0.8565
max_depth = 24 정확도 : 0.8565
```

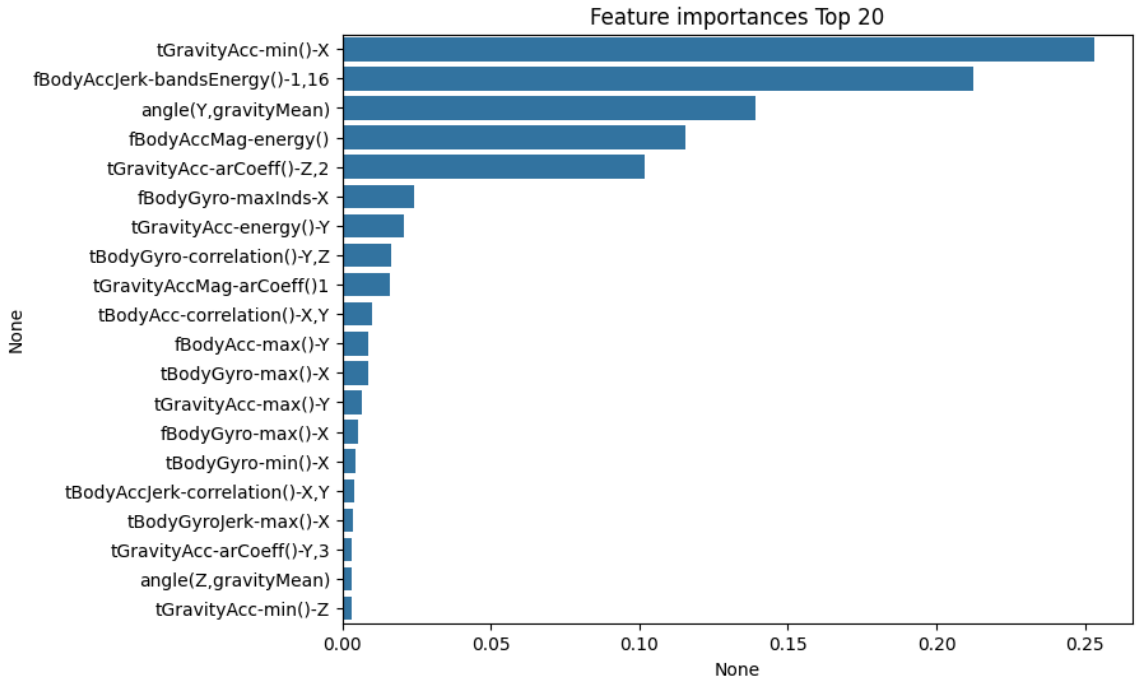
마찬가지 8일때 최대, 8 이후 정확도 감소 중

c. `max_depth`와 `min_samples_split` 같이 변경하며 정확도 성능 튜닝

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
GridSearchCV 최고 평균 정확도 수치 : 0.8549
GridSearchCV 최적 하이퍼 파라미터 :  {'max_depth': 8, 'min_samples_split': 16}
```

max_depth =8, min_samples_split = 16일 때, 85.4%

6. 각 피처 중요도 확인 - `feature_importances_` 속성



4.3 앙상블 학습

앙상블 학습 개요

> 💡 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합해
> 더 정확한 최종 예측을 도출하는 기법

- 이미지, 영상, 음성 등 비정형 데이터의 분류 : 딥러닝 **>** 앙상블
- 정형 데이터의 분류 : 딥러닝 **<** 앙상블

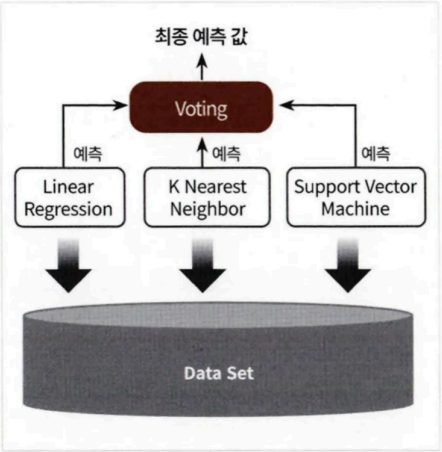
- 랜덤 포레스트, 그래디언트 부스트
  - 뛰어난 성능, 쉬운 사용, 다양한 활용도
- XGBoost
  - 캐글의 '매력적인 솔루션'이
- LightBGM
  - XGBoost 유사한 성능, 수행 속도 ↑
- Stacking
  - 여러 모델의 결과 기반 메타 모델 수립

라 불림

- 유형

▼ 보팅

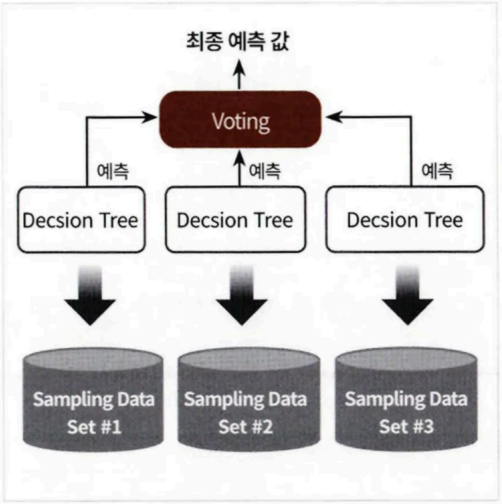
- 여러 개의 분류기가 투표를 통해 최종 예측 결과 결정
- 서로 **다른** 알고리즘을 가진 분류기 결합



Voting 방식

▼ 배깅

- 여러 개의 분류기가 투표를 통해 최종 예측 결과 결정
- 각 분류기가 **같은** 유형의 알고리즘 기반이지만, **데이터 샘플링을 서로 다르게** 가져감 = Bootstrapping 분할 방식
  - ex) 랜덤 포레스트
- 중첩 허용



Bagging 방식

▼ 부스팅

- 여러 분류기가 순차적으로 학습하되,

  앞에서 예측이 틀린 데이터를 다음에는 올바르게 예측할 수 있도록

  다음 분류기에게는 **가중치weight 부여** = 가중치 부스팅

▼ 스태킹

- 여러 다른 모델의 **예측 결과 → 다시 학습 데이터로,**

  다른 모델(메타 모델)로 재학습시켜 결과 예측

- ex) gradient boost, XGBoost, LightGBM

보팅 유형 - 하드 보팅 / 소프트 보팅

1. 하드 보팅

   예측값들 중 **다수의 분류기가 결정**한 예측값

   → 최종 보팅 결괏값

2. **소프트 보팅**

   (1)분류기들의 **레이블 값 결정 확률 다 더하고**

   (2)**평균**해서

   (3)이들 중 **확률이 높은** 레이블

   → 최종 보팅 결괏값

보팅 분류기(Voting Classifier)

위스콘신 유방암 데이터 세트 예측해보기

- 유방암의 악성종양 vs. 양성종양
- 크기, 모양 등 다양한 feature

- `VotingClassifier` 클래스
  - `estimators` 인자 : 리스트 형식으로 보팅에 사용될 여러 Classifier 객체들을 튜플 형식으로 입력
  - `voting` 인자 : '`hard`' 시 하드 보팅(Default), '`soft`'시 소프트 보팅

4.4 랜덤 포레스트

랜덤 포레스트 개요 및 실습

- 배깅의 대표 알고리즘
- 기반 알고리즘 : Decision Tree
- (+)빠르다, 쉽다, 직관적이다
- (-)하이퍼 파라미터가 많다, 그를 위한 튜닝 시간이 소모된다, 튜닝 후 성능이 크게 향상 드물다
- **Subset** 데이터 : 부트스트래핑 분할로 만들어진 임의의 데이터 세트들
- `n_estimators` : Subset 데이터 개수

랜덤 포레스트 하이퍼 파라미터, 튜닝

▼ 하이퍼 파라미터

- `n_estimators` : 결정 트리의 개수 (Defalut 10개)
  - 많이 설정할수록 좋은 성능을 기대할 수 있지만, 무제한 향상은 아님
  - 늘릴수록 수행 시간 ↑
- `max_features` : 결정 트리에 사용된 max_features 파라미터 (Default '`auto`'='`sqrt`')
  - 트리를 분할하는 피처를 참조할 때 얼만큼 참조할지
- `max_depth` : (DT와 동일)
- `min_samples_leaf` : (DT와 동일)
- `min_samples_split` : (DT와 동일)

- 최적 하이퍼 파라미터 튜닝

  1. GridSearchCV 수행

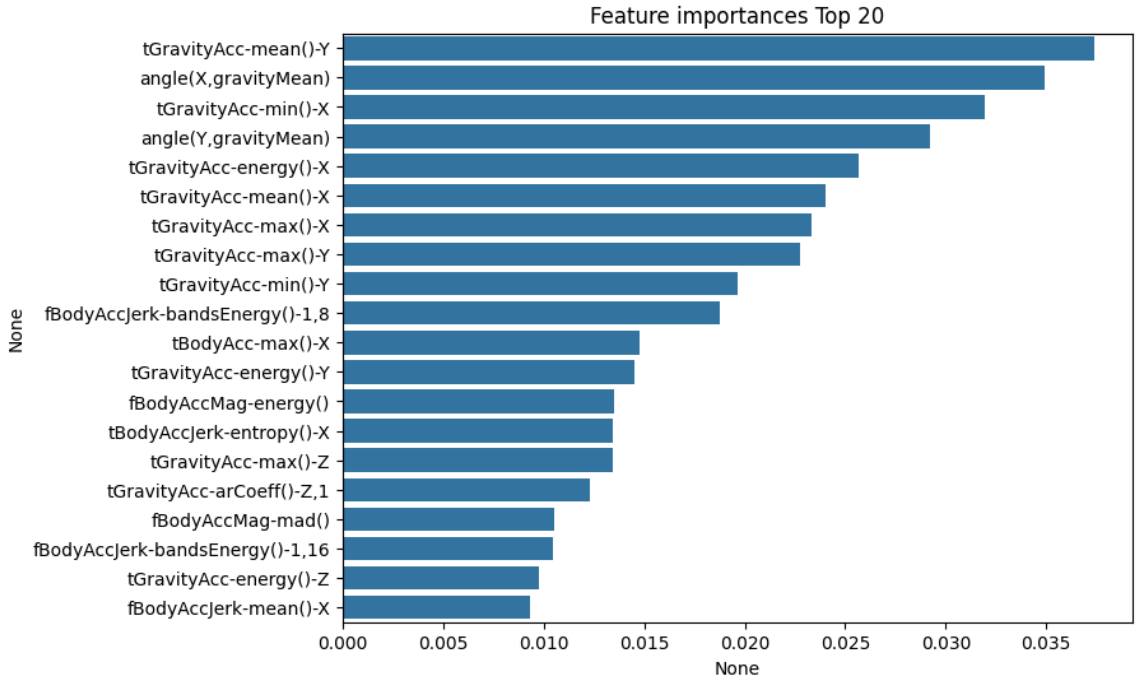
```
params = {
    'max_depth' : [8,16,24],
    'min_samples_leaf' : [1,6,12],
    'min_samples_split' : [2,8,16]
}
```

  2. 추출된 최적 하이퍼파라미터로 별도의 데이터 세트에서 성능 측정

```
rf_clf1 = RandomForestClassifier(n_estimators=100, min_samples_leaf=
6, max_depth=16,
                                 min_samples_split=16, random_state=0)

rf_clf1.fit(X_train, y_train)
pred = rf_clf1.predict(X_test)
print('예측 정확도 : {0:.4f}'.format(accuracy_score(y_test,pred)))
```

  3. `feature_importances_` 속성으로 피처 중요도 확인


Feature importances Top 20
tGravityAcc-mean()-Y
angle(X,gravityMean)
tGravityAcc-min()-X
angle(Y,gravityMean)
tGravityAcc-energy()-X
tGravityAcc-mean()-X
tGravityAcc-max()-X
tGravityAcc-max()-Y
tGravityAcc-min()-Y
fBodyAccJerk-bandsEnergy()-1,8
tBodyAcc-max()-X
tGravityAcc-energy()-Y
fBodyAccMag-energy()
tBodyAccJerk-entropy()-X
tGravityAcc-max()-Z
tGravityAcc-arCoeff()-Z,1
fBodyAccMag-mad()
fBodyAccJerk-bandsEnergy()-1,16
tGravityAcc-energy()-Z
fBodyAccJerk-mean()-X
None
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
None