04 분류

01 분류의 개요

- 베이즈(Bayes) 통계와 생성 모델에 기반한 나이브 베이즈(Naive Bayes)
- 독립변수와 종속변수의 선형 관계성에 기반한 로지스틱 회귀(Logistic Regression)
- 데이터 균일도에 따른 규칙 기반의 결정 트리(Decision Tree)
- 개별 클래스 간의 최대 분류 마진을 효과적으로 찾아주는 서포트 벡터 머신(Support Vector Machine)
- 근접 거리를 기준으로 하는 최소 근접(Nearest Neighbor) 알고리즘
- 심층 연결 기반의 신경망(Neural Network)
- 서로 다른 머신러닝 알고리즘을 결합한 앙상블(Ensemble)

앙상블 방법(Ensemble Method)

서로 다른/또는 같은 알고리즘을 결합하며 대부분은 **동일한 알고리즘**을 결합한다.

앙상블의 기본 알고리즘으로 일반적으로 사용하는 것은 **결정 트리**이다.

배깅(Bagging)

랜덤 포레스트(Random Forest) : 뛰어난 예측 성능, 상대적으로 빠른 수행 시간, 유연성

부스팅(Boosting)

그래디언트 부스팅(Gradient Boosting) : 뛰어난 예측 성능, 그러나 **수행 시간이 오래 걸림**

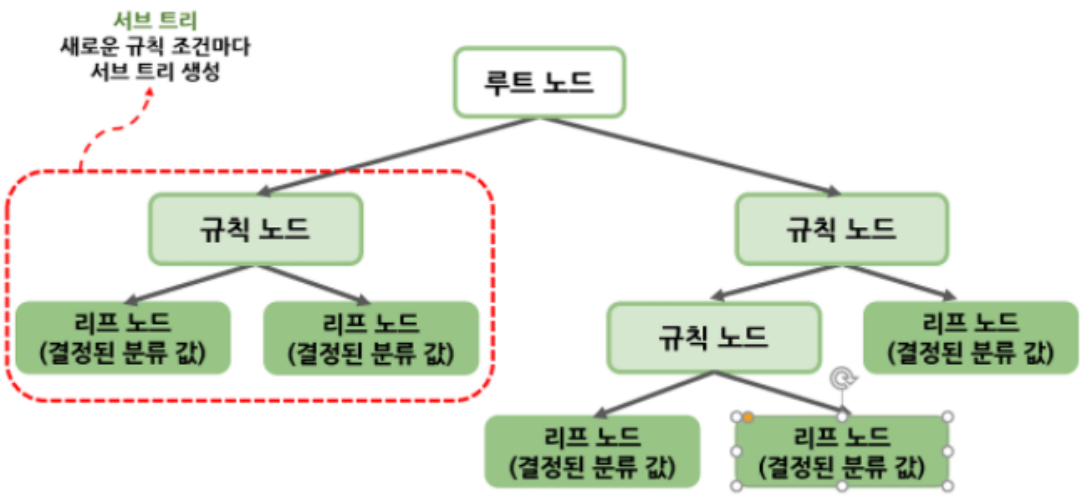
XgBoost, LightGBM : 기존 그래디언트 부스팅의 예측 성능을 한 단계 발전시키면서도 수행 시간을 단축 시킴

02 결정 트리

결정 트리는 ML 알고리즘 중 **직관적으로 이해하기 쉬운 알고리즘**이다

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

데이터의 어떤 기준을 바탕으로 규칙을 만들어야 가장 효율적인 분류가 될 것인가가 알고리즘의 성능을 크게 좌우한다



결정 트리의 구조

규칙 노드 : 규칙 조건, 리프 노드 : 결정된 클래스 값, 새로운 규칙 조건마다 서브 트리가 생성

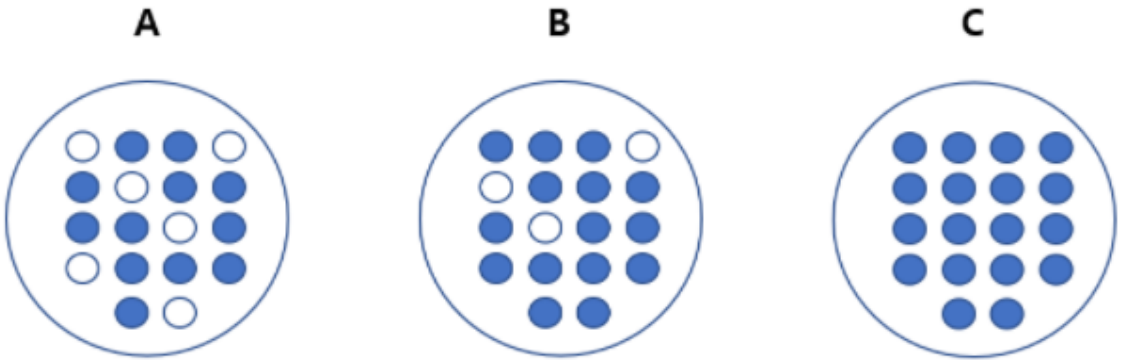
데이터 세트에 피처가 있고 이러한 피처가 결합해 규칙 조건을 만들 때마다 규칙 노드가 만들어진다.

하지만 **많은 규칙이 있다는 것은 곧 분류를 결정하는 방식이 더욱 복잡해지며 과적합으로 이어지기 쉽다.**

즉, 트리의 깊이가 깊어질수록 결정트리의 예측 성능이 저하될 가능성이 높다.

가능한 한 적은 결정 노드로 높은 예측 정확도를 가지려면 데이터를 분류할 때 최대한 많은 데이터 세트가 해당 분류에 속할 수 있도록 결정 노드의 규칙이 정해져야 한다

이를 위해선 최대한 **균일한 데이터 세트**를 구성할 수 있도록 분할하는 것이 필요하다



균일도 : C>B>A 이며 이러한 데이터 세트의 균일도는 데이터를 구분하는데 필요한 **정보의 양**에 영향을 미친다.

결정 노드는 **정보 균일도가 높은 데이터 세트를 먼저 선택**할 수 있도록 규칙 조건을 만든다

이러한 정보의 균일도를 측정하는 대표적인 방법은 **엔트로피를 이용한 정보 이득 지수**와 **지니 계수**가 있다.

- 정보 이득은 엔트로피라는 개념을 기반으로 한다. 엔트로피는 주어진 데이터 집합의 혼잡도를 의미하는데, 서로 다른 값이 섞여 있으면 엔트로피가 높고, 같은 값이 섞여 있으면 엔트로피가 낮다. 정보 이득 지수는 1에서 엔트로피 지수를 뺀 값이다. 즉, 1 - 엔트로피 지수이다. 결정 트리는 이 정보 이득 지수로 분할 기준을 정한다. 즉, 정보 이득이 높은 속성을 기준으로 분할한다.
- 지니 계수는 원래 경제학에서 불평등 지수를 나타낼 때 사용하는 계수이다. 0이 가장 평등하고 1로 갈수록 불평등하다. 머신러닝에 적용될 때는 지니 계수가 낮을 수록 데이터의 균일도가 높은 것으로 해석해 지니 계수가 낮은 속성을 기준으로 분할한다.

1. 데이터 집합의 모든 아이템이 같은 분류에 속하는지 확인한다
    - If true : 리프 노드로 만들어서 분류를 결정한다
    - Else : 데이터를 분할하는 데 가장 좋은 속성과 분할 기준을 찾는다.(정보 이득, 지니계수 이용)
2. 해당 속성과 분할 기준으로 데이터 분할하여 Branch 노드를 생성한다.
3. Recursive하게 모든 데이터 집합의 분류가 결정될 때 까지 위 과정을 반복한다.

결정 트리 모델의 특징

결정 트리의 가장 큰 장점

- **정보의 '균일도'라는 룰을 기반으로 하고 있어서 알고리즘이 쉽고 직관적이다.**

- 룰이 매우 명확하고, 어떻게 규칙 노드와 리프 노드가 만들어지는지 알 수 있고, 시각화로 표현까지 할 수 있다.
- 각 피처의 스캐일링과 정규화 같은 전처리 작업이 필요없다.

결정 트리의 가장 큰 단점

- **과적합으로 정확도가 떨어진다. 이를 극복하기 위해 트리의 크기를 사전에 제한하는 튜닝이 필요하다.**

결정 트리 파라미터

`DecisionTreeClassifier` **: 분류를 위한 클래스**

`DecisionTreeRegreesor` **: 회귀를 위한 클래스**

`min_sample_split` : 노드를 분할하기 위한 최소한의 샘플 데이터 수

`min_sample_leaf` : 말단 노드가 되기 위한 최소한의 샘플 데이터 수

`max_features` : 최적의 분할을 위해 고려할 최대 피처 개수. 디폴트는 None으로 데이터 세트의 모든 피처를 사용해 분할 수행

`max_depth` : 트리의 최대 깊이를 규정.

`max_leaf_nodes` : 말단 노드의 최대 개수

결정 트리 모델의 시각화

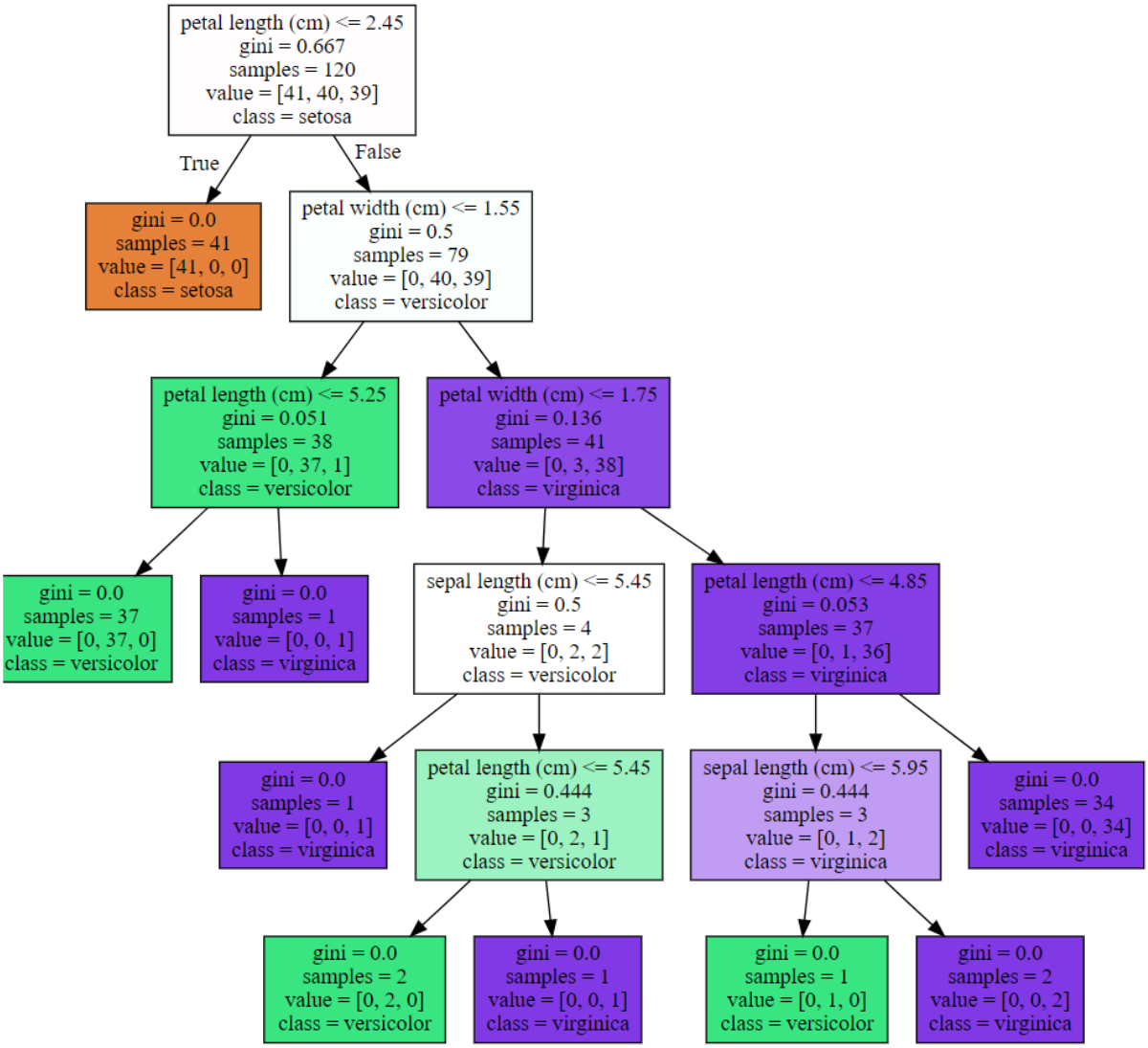
Graphviz 패키지 사용 : 결정 트리 알고리즘이 어떠한 규칙을 가지고 트리르 생성하는 지 시각적으로 보여줄 수 있음

`export_graphviz()` : 학습이 완료된 Estimator, 피처의 이름 리스트, 레이블 이름 리스트를 입력하면 학습된 결정 트리 규칙을 실제 트리 형태로 시각화해 보여준다.

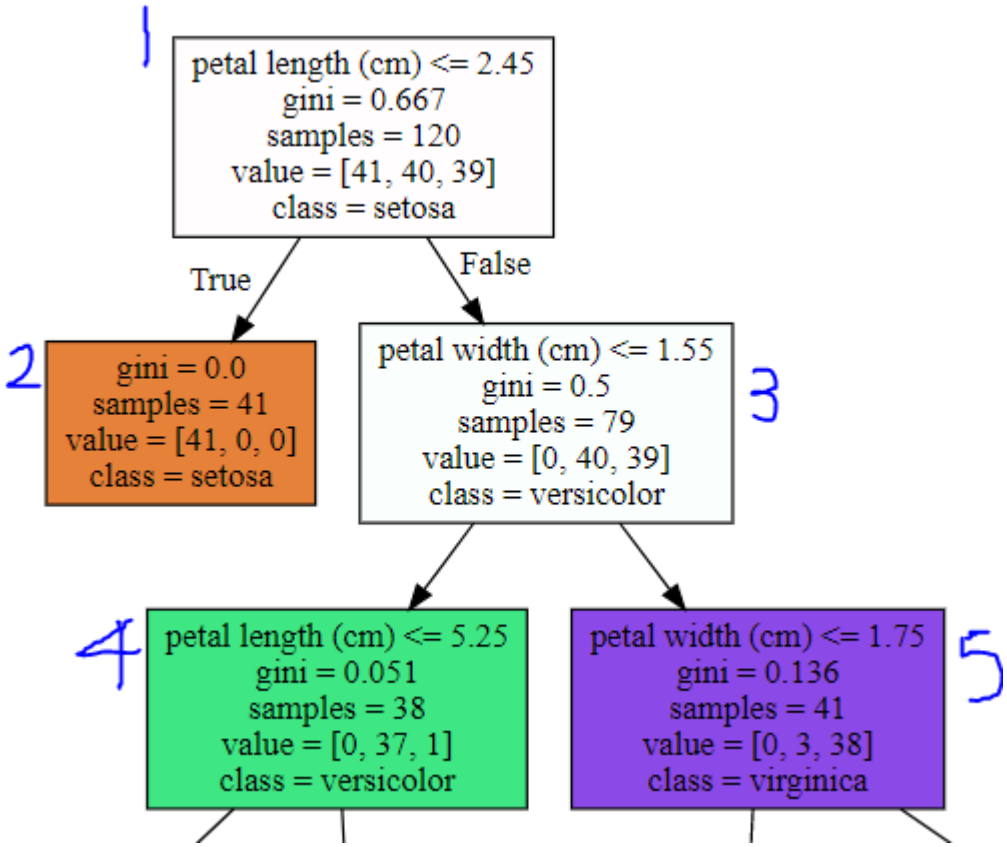
p. 191 impurity, filled

Graphviz를 이용해 붓꽃 데이터 세트에 결정트리 적용

각 노드의 색깔은 붓꽃 데이터의 레이블 값이다. 주황색은 Setosa, 초록색은 Versicolor, 보라색은 Virginica 레이블을 나타낸다. 색깔이 짙어질 수록 지니계수가 낮고, 해당 레이블에 속하는 샘플 데이터가 많다는 의미이다.



- `petal length(cm) ≤ 2.45`와 같이 피처의 조건이 있는 것은 자식 노드를 만들기 위한 규칙 조건이다. 이 조건이 없으면 리프 노드이다.\
- `gini`는 다음의 `value = []`로 주어진 데이터 분포에서의 지니 계수이다.
- `samples`는 현 규칙에 해당하는 데이터 건수이다.
- `value  = []`는 클래스 값 기반의 데이터 건수이다. 붓꽃 데이터 세트는 클래스 값으로 0,1,2를 가지고 있으며, 0: Setosa, 1: Versicolor, 2: Viginica 품종을 말한다. 만일 `Value = [41,40,39]`라면 클래스 값의 순서로 Setoda 41개, Versicolor 40개, Viginica 39개로 데이터가 구성돼 있다는 의미이다.



1번 노드의 지표 설명

- `samples = 120`개는 전체 데이터가 120개라는 의미이다.
- `value = [41,40,39]`는 Setosa 41개, Versicolor 40개, Virginica 39개로 데이터 구성
- sample 120개가 `value = [41,40,39]` 분포도로 되어 있으므로 지니 계수는 0.667
- `petal length(cm) ≤ 2.45` 규칙으로 자식 노드 생성
- `class = setosa`는 하위 노드를 가질 경우에는 setosa의 개수가 41개로 제일 많다는 의미

petal length(cm) ≤ 2.45로 True/False로 분기

2번 노드의 지표 설명

- 41개의 샘플 데이터 모두 Setosa이므로 예측 클래스는 Setosa로 결정
- 지니 계수는 0임

3번 노드의 지표 설명

- 79개의 샘플 데이터 중 Versicolor 40개, Virginica 39개로 여전히 지니 계수는 0.5로 높으므로 **다음 자식 브랜치 노드로 분기할 규칙이 필요하다**
- `petal length(cm) ≤ 1.55` 규칙으로 자식 노드 생성

4번 노드의 지표 설명

- 38개의 샘플 데이터 중 Versicolor 37개 Virginica 1개로 대부분이 Versicolor임
- 지니 계수는 0.051로 매우 낮으나, 여전히 Versicolor와 Virginica가 혼재되어 있으므로 `petal length(cm) ≤ 5.25`라는 새로운 규칙으로 다시 자식 노드 생성

5번 노드의 지표 설명

- 41개의 샘플 데이터 중 Versicolor가 3개, Virginica가 38개로 대부분이 Virginica임
- 지니 계수는 0.136으로 낮으나 여전히 Versicolor와 Viginica가 혼재되어 있으므로, `petal length(cm) ≤ 1.75`라는 새로운 규칙으로 다시 자식 노드 생성

4번 노드를 보면 1개의 데이터를 구분하기 위해 다시 자식 노드를 생성한다. 이처럼 결정 트리는 규칙 생성 로직을 미리 제어하지 않으면 완벽하게 클래스 값을 구별해내기 위해 트리 노드를 계속해서 만들어간다. 결국, **매우 복잡한 규칙 트리가 만들어져** 모델이 쉽게 **과적합**되는 문제점을 가지게 된다.

이 때문에 결정 트리 알고리즘을 제어하는 대부분 **하이퍼 파라미터**는 **복잡한 트리가 생성되는 것을 막기위한 용도**이다.

`max_depth` **하이퍼 파라미터**로 변경했을 시 트리 깊이가 제한되기 때문에 더 간단한 결정트리가 된다.

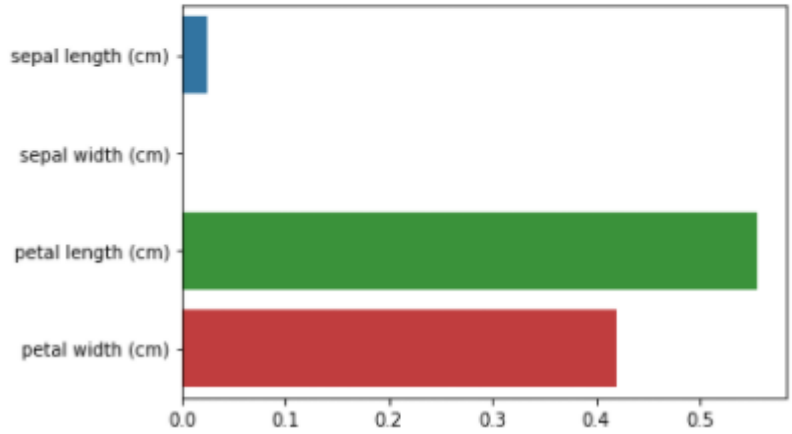
`min_samples_splits` **하이퍼 파라미터**로 변경했을 시 최소한의 샘플 데이터 개수를 제한하기 때문에 `min_samples_splits =4`, 즉 자식 노드로 분할할 때, 트리 깊이를 줄일 수 있다.

`min_samples_leafs` **하이퍼 파라미터**로 리프 노드가 될 수 있는 최소한의 샘플 데이터를 설정하기 때문에 리프 노드가 될 수 있는 조건이 완화되어 트리 깊이를 줄일 수 있다.

결정트리는 균일도에 기반해 **어떠한 속성을 규칙 조건으로 선택하느냐**가 중요한 요건이다.

`feature_importances_`는 피처의 중요한 역할 지표를 속성으로 제공하며 결정 트리 알고리즘이 어떻게 동작하는지 직관적으로 이해할 수 있다.

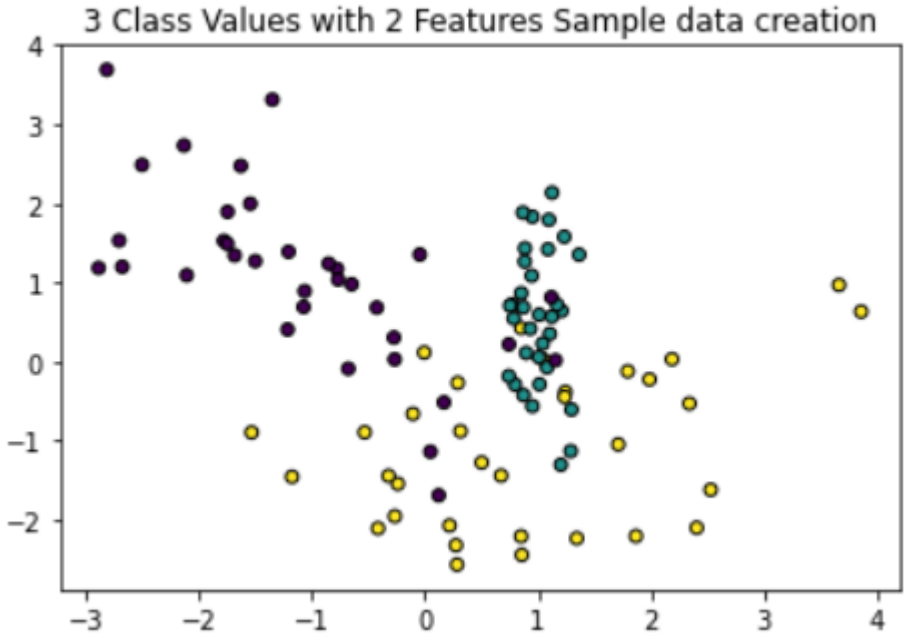
```
Feature importance:
[0.025 0.    0.555 0.42 ]
sepal length (cm) : 0.025
sepal width (cm) : 0.000
petal length (cm) : 0.555
petal width (cm) : 0.420

<AxesSubplot:>
```



여러 피처들 중 petal_length가 가장 피처 중요도가 높음을 알 수 있다.

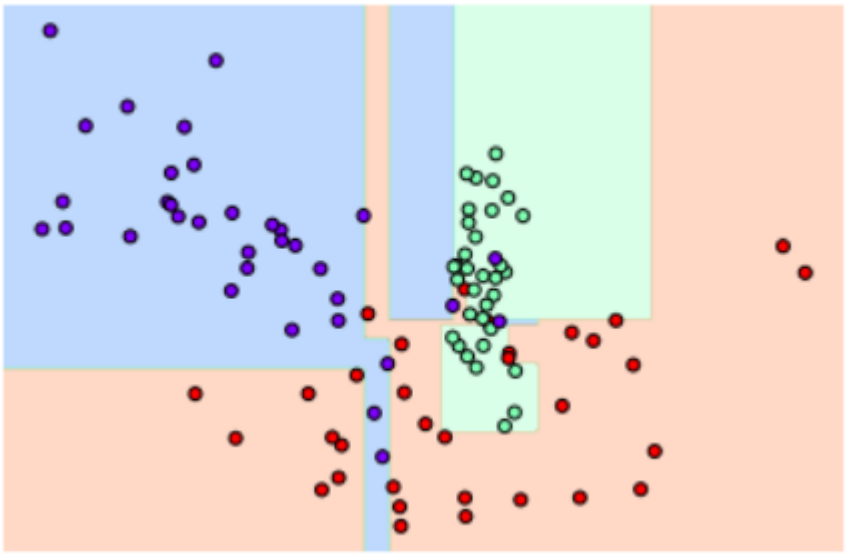
결정 트리 과적합(Overfitting)

각 피처가 X, Y 축으로 나열된 2차원 그래프이며, 3개의 클래스 값 구분은 색깔로 되어있다.



결정 트리 모델이 어떠한 결정 기준을 가지고 분할하면서 데이터를 분류하는 지 확인하자

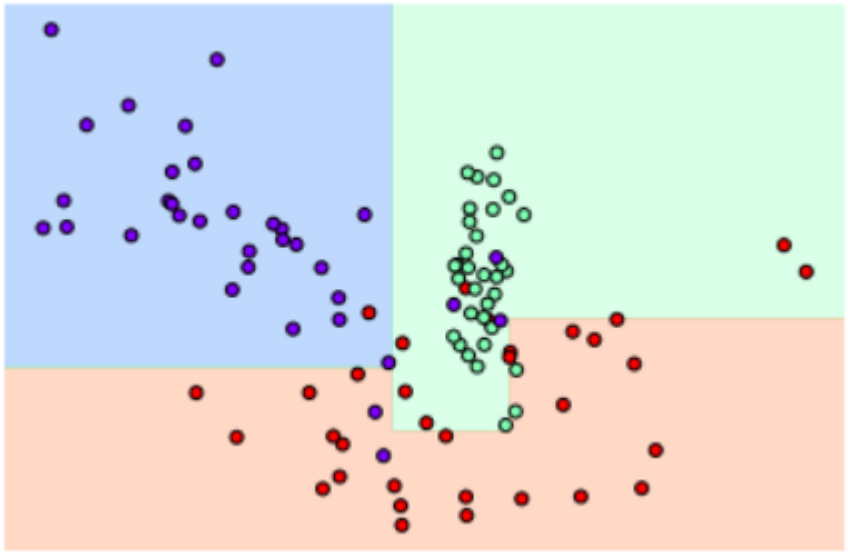
`visualize_boundary()` : 머신러닝 모델이 클래스 값을 예측하는 결정 기준을 색상과 경계로 나타내 모델이 어떻게 데이터 세트를 예측 분류하는지 잘 이해할 수 있게 해준다.



일부 이상치 데이터 까지 분류하기 위해 분할이 자주 일어나 결정 기준 경계가 매우 많아졌다.

이는 **학습 데이터 세트의 특성과 약간만 다른 형태의 데이터 세트를 예측**하면 **예측 정확도가 떨어진다.**

이번에는 min_samples_leaf = 6을 설정해 6개 이하의 데이터는 리프 노드를 생성할 수 있도록 하자.



이상치에 크게 반응하지 않으면서 좀 더 일반화된 분류 규칙에 따라 분류되었음을 알 수 있다. 따라서 **학습 데이터에만 지나치게 최적화된 분류기준**은 오히려 **테스트 데이터 세트**에서

정확도를 떨어뜨릴 수 있다.

결정 트리 실습 - 사용자 행동 인식 데이터 세트

Human_activity 디렉터리 불러오기

중복된 피처명 제거

```
column_index    42
dtype: int64
```

column_name	column_index
fBodyAcc-bandsEnergy()-1,16	3
fBodyAcc-bandsEnergy()-1,24	3
fBodyAcc-bandsEnergy()-1,8	3
fBodyAcc-bandsEnergy()-17,24	3
fBodyAcc-bandsEnergy()-17,32	3

총 42개의 피처명이 중복되어 있다. 이 중복된 피처명에 대해서는 _1 또는 _2를 추가해 새로운 피처명을 가지는 DataFrame을 반환하는 함수인 get_new_feature_name_df()를 생성한다.

같은 index가 3개씩 중복되서 결국 42개가 남는다.

get_human_dataset()로 피처 데이터 파일과 레이블 데이터 파일을 각각 학습/테스트용 DataFrame에 로드하자

로드한 학습용 피처 데이터 세트를 간략히 살펴보자

```
## 학습 피처 데이터 셋 info()
<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
None
```

학습 데이터 세트는 7352개의 레코드로 561개의 피처를 가지고 있으며 많은 칼럼의 대부분이 움직임 위치와 관련된 속성임을 알 수 있다.

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

5 rows × 561 columns

레이블 데이터 세트는 고르게 분포되어 있음을 확인할 수 있다.

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

DecisionTreeClassifier의 하이퍼 파라미터는 모두 디폴드 값으로 설정해 수행하고, 이 때의 하이퍼 파라미터 값을 모두 추출해보자

```
결정 트리 예측 정확도 : 0.8548
DecisionTreeClassifier 기본 하이퍼 파라미터:
 {'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, 'random_state': 156, 'splitter': 'best'}
```

약 85.48%의 정확도를 나타내고 있다.

이번에는 결정 트리의 트리 깊이가 예측 정확도에 주는 영향을 살펴보자. GridSearchCV를 이용해 하이퍼 파라미터인 max_depth 값을 변화시키면서 예측 성능을 확인해보자.

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
GridSearchCV 최고 평균 정확도 수치: 0.8513
GridSearchCV 최적 하이퍼 파라미터: {'max_depth': 16}
```

	param_max_depth	mean_test_score
0	6	0.850791
1	8	0.851069
2	10	0.851209
3	12	0.844135
4	16	0.851344
5	20	0.850800
6	24	0.849440

max_depth가 8일 때 0.852로 정확도가 정점이고, 이를 넘어가면서 깊어진 트리는 검증 데이터 세트에서 오히려 과적합으로 인한 성능 저하를 유발한다.

이번에는 별도의 테스트 데이터 세트에서 결정 트리의 정확도를 측정해보자.

```
max_depth = 6 정확도 :0.8558
max_depth = 8 정확도 :0.8707
max_depth = 10 정확도 :0.8673
max_depth = 12 정확도 :0.8646
max_depth = 16 정확도 :0.8575
max_depth = 20 정확도 :0.8548
max_depth = 24 정확도 :0.8548
```

max_depth가 8인 경우, 정확도가 87.07%로 가장 높다.

max_depth와 min_samples_split을 같이 변경하면서 정확도 성능을 튜닝해보자.

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

별도 분리된 테스트 데이터 세트에 해당 하이퍼 파라미터를 적용해보자. `best_estimator_`는 최적 하이퍼 파라미터로 학습이 완료된 Estimator 객체이다.

결정 트리 예측 정확도 : 0.8717%임을 확인할 수 있다.

03 앙상블 학습

앙상블 학습 개요

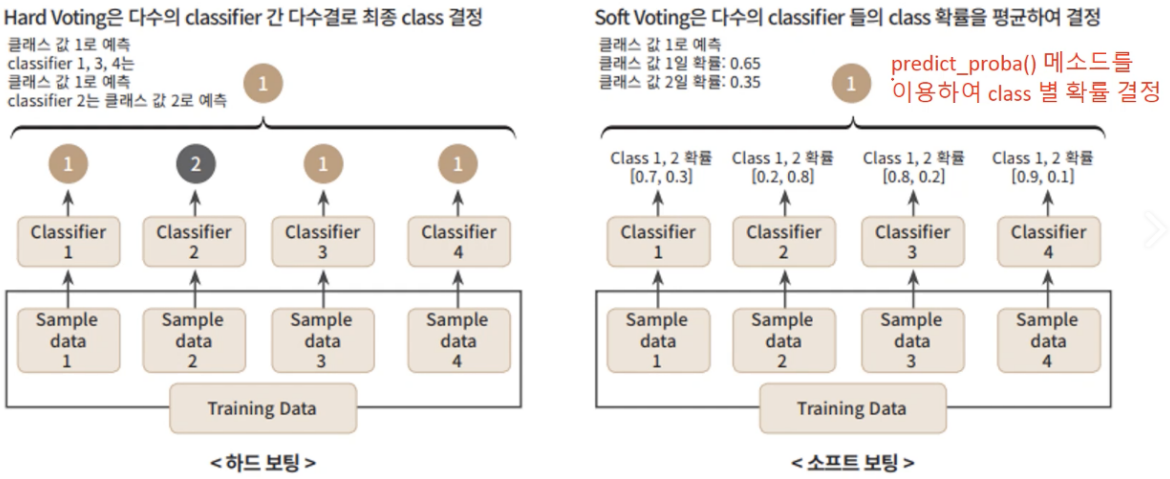
앙상블 학습을 통한 분류 : 여러 개의 분류기를 생성하고 그 예측을 결합함으로써 보다 정확한 최종예측을 도출하는 기법

정형 데이터 분류 시 뛰어난 성능을 나타내고 있다.

앙상블 학습의 유형

- 보팅 : **서로 다른 알고리즘**을 가진 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정한다.
- 배깅 : 같은 유형의 알고리즘을 가진 여러 개의 분류기가 **데이터 샘플링을 서로 다르게 가져가면서** 투표를 통해 최종 예측 결과를 결정한다. 대표적인 배깅방식이 랜덤 포레스트이다. 개별 Classifier에 데이터를 샘플링해서 추출하는 부트스트래핑 분할 방식을 이용한다.
- 부스팅 : 여러 분류기가 순차적으로 학습을 수행하되, 예측이 틀린 데이터에 대해서는 올바르게 예측할 수 있도록 **다음 분류기에는 가중치를 부여**하면서 학습과 예측을 진행하는 방식이다.
- 스태킹 : 여러 가지 다른 모델의 예측 결과값을 다시 학습데이터로 만들어서 다른 모델로 재학습시켜 결과를 예측하는 방법이다.

보팅 유형 - 하드 보팅과 소프트 보팅



하드 보팅은 예측한 결과값들 중 다수의 분류기가 결정한 예측값을 최종 보팅 결과값으로 선정하는 것이다.

소프트 보팅은 분류기의 레이블 결정값들을 평균해서 이들 중 가장 확률이 높은 레이블 값을 최종 보팅 결과값으로 선정한다.

보팅 분류기

보팅 방식의 앙상블을 이용해 위스콘신 유방암 데이터 세트를 예측 분석에 이용하자.

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	wors textur
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.3
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.4
2	19.69	21.25	130.0	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.5

로지스틱 회귀와 KNN을 기반으로 하여 소프트 보팅 방식으로 새롭게 보팅 분류기를 만들어 보자.

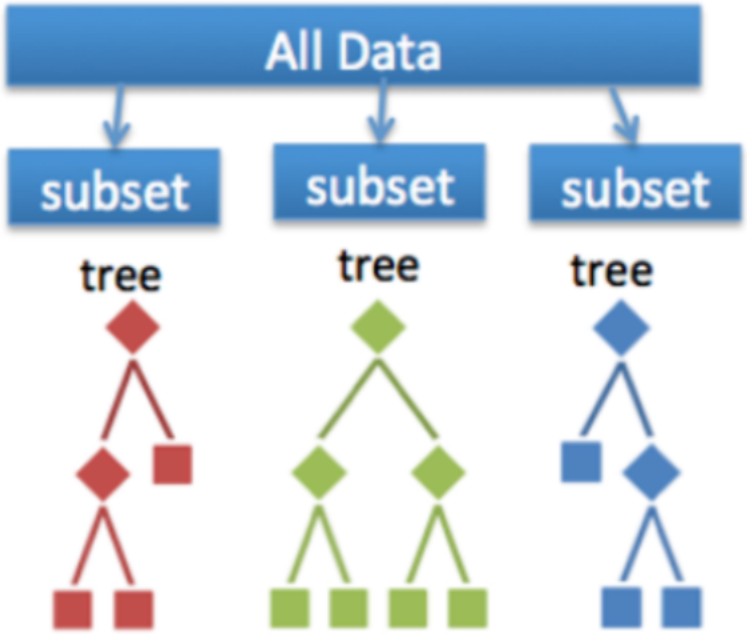
```
Voting 분류기 정확도 : 0.9474
LogisticRegression 정확도 : 0.9386
KNeighborsClassifier 정확도 : 0.9386
```

보팅 분류기의 정확도가 높게 나타남을 알 수 있다.

ML 모델의 성능은 이렇게 다양한 테스트 데이터에 의해 검증되므로 높은 유연성을 가지고 현실에 대처할 수 있는가가 중요한 ML 모델의 평가요소가 된다. 이런 관점에서 편향-분산 트레이드 오프는 극복해야할 문제이다. 앙상블 학습에서는 결정 트리 알고리즘의 단점을 매우 많은 분류기를 통해서 극복할 수 있다. 따라서 결정 트리 알고리즘의 장점은 그대로 취하고 편향-분산 트레이드 오프의 효과를 극대화할 수 있다.

04 랜덤 포레스트

랜덤 포레스트의 개요 및 실습



배깅은 **같은 알고리즘으로 여러 개의 분류기를 만들어서** 보팅으로 최종 결정하는 알고리즘이다.

랜덤 포레스트 : 배깅의 대표적인 알고리즘으로, 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 **각자의 데이터를 샘플링**해 개별적으로 학습을 수행한 뒤 최종적으로 모든 분류기가 보팅을 통해 예측 결정을 하게 된다.

< 부트스트래핑 샘플링 방식 >

여러 개의 데이터 세트를 중첩되게 분리하는 방식인 **부트스트래핑**을 통해 데이터를 분리하고 중첩된 개별 데이터 세트에 결정트리 분류기를 각각 적용하는 것을 **랜덤 포레스트**라고 한다.

랜덤 포레스트 정확도: 0.9253

랜덤 포레스트는 사용자 행동 인식 데이터 세트에 대해 약 92.53%의 정확도를 보여준다.

랜덤 포레스트 하이퍼 파라미터 튜닝

트리 기반의 앙상블 파라미터의 단점

- 하이퍼 파라미터라 너무 많다
- 튜닝을 위한 시간이 너무 많이 소요됨

랜덤 포레스트의 하이퍼 파라미터

- `n_estimators` : 랜덤 포레스트에서 결정 트리의 개수를 지정한다. 디폴트는 10개이다. 많이 설정할 수록 좋은 성능을 기대할 수 있지만 무조건적인 성능 향상은 바랄 수 없으며, 수행 시간이 늘어난다.
- `max_features` : 결정 트리에 사용된 `max_features` 파라미터와 같다. 하지만, `RandomForestClassifier`의 기본 `max_features` 는 'None'이 아니라 'Auto', 즉 'sqrt'와 같다.
- `max_depth` 나 `min_samples_leaf` 와 같이 결정 트리에서 과적합을 개선하기 위해 사용되는 파라미터가 랜덤 포레스트에도 똑같이 적용될 수 있다.

GridSearchCV를 이용한 랜덤 포레스트의 하이퍼 파라미터 튜닝

- n_estimator를 100 cv를 2로 설정해 하이퍼 파라미터를 구하자

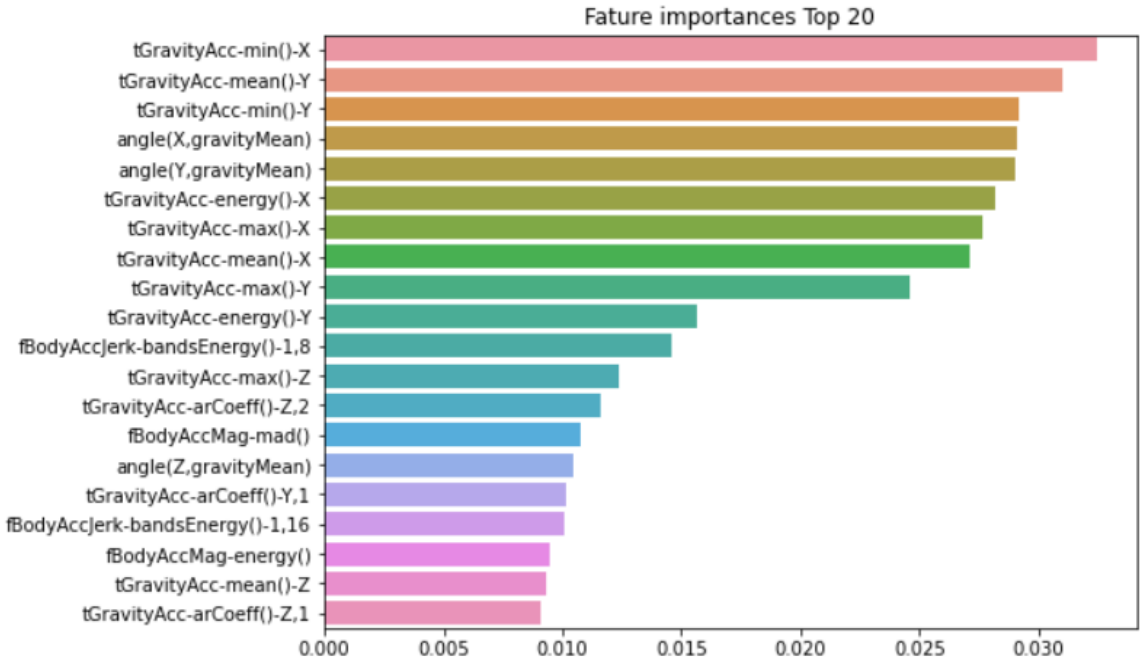
```
{'max_depth': 10, 'min_samples_leaf': 8, 'min_samples_split': 8, 'n_estimators': 100}
0.9179815016322089
```

- n_estimator를 300으로 증가시켜서 다시 RandomForestClassifier를 이용해 별도의 테스트 데이터에서 예측을 수행하자.

```
0.9165252799457075
```

즉, estimator를 많이 설정한다고 해서 **항상 좋은 성능이 나오는 것은 아님**을 알 수 있다.

features_importance_ 속성을 이용해 피처의 중요도를 구하자.



05 GBM(Gradient Boosting Machine)

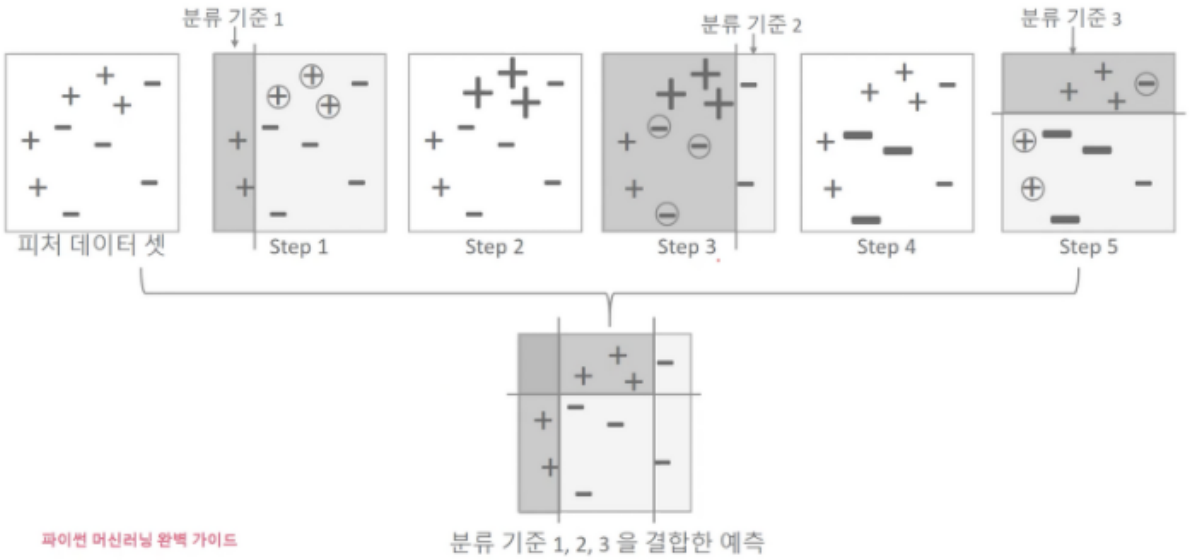
GBM의 개요 및 실습

여러 개의 약한 학습기를 순차적으로 학습 예측하면서 잘못 예측한 데이터에 가중치를 부여해 오류를 개선하면서 학습하는 방식이다.

부스팅의 대표적인 구현

1. AdaBoost

AdaBoost : 오류 데이터에 가중치를 부여하면서 부스팅을 수행하는 알고리즘



1. 첫 번째 약한 학습기가 분류 기준 1로 +와 -를 분류한다.
2. + 오류 데이터에 가중치 값을 부여해서, 가중치가 부여된 오류 + 데이터에는 다음 약한 학습기가 더 잘 분류할 수 있게 커졌다.
3. 두 번째 약한 학습기가 기준 2로 분류하고 오류 - 데이터에 가중치를 부여했다.
4. 세 번째 약한 학습기가 기준 3으로 분류를 진행하고 오류 데이터를 찾는다.
5. 예측 결정 기준을 모두 결합해 예측을 수행한다.

GBM(Gradient Boost Machine)

GBM : 반복 수행을 통해 오류를 최소화할 수 있도록 **가중치의 업데이트 값을 도출하는 기법**이다.

가중치 업데이트는 **경사 하강법**을 이용한다.

오류 값은 실제 값 - 예측 값이므로, 분류의 실제 결과값을 y , 피처를 $x_1, x_2, x_3, ..., x_n$, 그리고 이 피처에 기반한 예측 함수를 $F(x)$ 함수라고 하면 오류식은 $h(x) = y - F(x)$ 이 된다.

GradientBoostingClassifier를 이용해 사용자의 행동 데이터 세트를 예측/분류 해보자.

기본 하이퍼 파라미터만으로 예측정확도가 랜덤포레스트보다 나음을 알 수 있다. 그러나, 수행시간이 오래 걸린다는 점, 하이퍼 파라미터의 튜닝을 해야한다는 점이 단점이다.

GBM 하이퍼 파라미터 튜닝

GBM 기반 파라미터

- loss : 경사 하강법에서 사용할 비용 함수를 지정한다.
- learning_rate : GBM이 학습을 진행할 때 마다 적용하는 학습률이다. Weak learner가 순차적으로 오류 값을 보정해 나가는 데 적용하는 계수이다. 기본값은 0.1이며, 너무 작은 값을 적용하면 업데이트되는 값이 작아져서 최소 오류값을 찾아 예측 성능이 높아질 가능성이 높지만 시간이 오래 걸린다. 반대로 큰 값을 적용하면 최소 오류값을 찾지못하고 그냥 지나쳐 예측 성능이 떨어질 가능성이 높지만, 빠른 수행이 가능하다.

  이러한 특성으로 n_estimator와 상호 보완적으로 조합해 사용한다.
- n_estimator : weak learner의 개수이다. 개수가 많을수록 예측 성능이 높아지지만 시간이 오래걸린다.
- subsample : weak learner가 학습에 사용하는 데이터의 샘플링 비율이다. 기본값은 1이며, 전체 학습 데이터를 기반으로 학습한다.

GridSearchCV를 이용해 하이퍼 파라미터 튜닝을 진행하자. n_estimator를 100, 500으로 learning_rate를 0.05, 0.1로 제약하고 교차 폴드 검증 세트를 2개로 제한하자.

책을 통해, learning_rate가 0.05, n_estimator가 500일 때 2개의 교차 검증세트에서 정확도가 90.1%로 최고를 기록했음을 알 수 있다.

책에서 테스트 데이터 세트에 적용했을 때 높은 정확도를 기록했다. 그러나, **수행시간이 오래걸리는 단점**이 있다.

06 XGBoost(eXtra Gradient Boost)

XGBoost 개요

- 트리 기반의 앙상블 학습에서 가장 각광받고 있는 알고리즘
- GBM의 단점인 **느린 수행 시간 및 과적합 규제 부재** 등의 문제를 해결함
- 뛰어난 예측 성능, GBM 대비 빠른 수행시간, 과적합 규제, Tree pruning, 자체 내장된 교차 검증, 결손값 자체 처리

파이썬 래퍼 XGBoost 적용 - 위스콘신 유방암 예측

`xgboost`(XGBoost의 파이썬 패키지) : 교차 검증, 성능 평가, 피처 중요도 등의 시각화 기능을 가지고 있다.

위스콘신 유방암 데이터 세트는 다양한 속성값을 기반으로 악성인지 양성 인지를 분류한 데이터 세트이다.

`XGBoost`는 학습용 데이터와 테스트용 데이터 세트를 위해 별도의 객체인 **DMatrix**를 생성한다.

DMatrix는 주로 넘파이 입력 파라미터를 받아 만들어지는 XGBoost만의 전용 데이터 세트이다. 주요 입력 파라미터는 data와 label이다.

XGBoost의 하이퍼 파라미터를 설정한다. 지정된 하이퍼 파라미터로 XGBoost 모델을 학습시키자.

학습 시 XGBoost는 수행 속도를 개선하기 위해서 **조기 중단 기능**을 제공한다.

조기 중단은 num_boost_round를 모두 채우지 않고 중간에 반복을 빠져나올 수 있도록 하는 것.

`eval_set` 는 성능 평가를 수행할 평가용 데이터 세트를 설정한다.

`eval_metirc` 은 평가 세트에 적용할 성능 평가 방법. 분류일 경우 주로 'error', 'logloss'를 적용한다.

`xgboost`의`predict()` 는 예측 결괏값이 아닌 예측 결과를 수정할 수 있는 **확률 값**을 반환한다.

사이킷런 래퍼 XGBoost의 개요 및 적용

사이킷런의 기본 `Estimator` 를 그대로 상속해 만들었기 때문에, 다른 `Estimator` 와 동일하게 `fit()`과 `predict()`만으로 학습과 예측이 가능하다.

파이썬 래퍼 XGBoost와 사이킷런 XGBoost의 하이퍼 파라미터에 약간의 차이가 존재한다.

`XGBoost`의 `n_estimator` 와 `num_boost_round` 하이퍼 파라미터는 서로 동일한 파라미터이다.

위스콘신 대학병원의 유방암 데이터 세트를 `XGBClassifier`를 이용해서 예측하자.

→ 파이썬 래퍼 `XGBoost`와 동일한 평가 결과가 나옴을 알 수 있다.

사이킷런 `XGBoost`도 조기 중단 기능을 수행할 수 있다.

1. `early_stopping_rounds`를 100으로 설정한 경우

- 100번의 반복 동안 성능 평가 지수가 향상되지 않았기 때문에, 더 이상 반복하지 않고 멈춘다.

2. `early_stopping_rounds`를 100으로 설정한 경우

- 10번의 반복 동안 성능평가 지수가 향상되지 못해, 더 이상의 반복을 수행하지 않고 종료된다.

07 LightGBM

`XGBoost`는 매우 뛰어난 부스팅 알고리즘이지만, 여전히 **시간이 오래 걸린다.**

`LightGBM` 의 가장 큰 장점

- **학습에 걸리는 시간이 훨씬 적다.**
- **메모리 사용량도 상대적으로 적다.**
- **카테고리형 피처의 자동변환과 최적 분할**

`LightGBM`은 일반 GBM 계열의 트리 분할 방법과 다르게 **리프 중심의 트리 분할 방식**을 사용한다.

08 분류 실습 - 캐글 산탄데르 고객 만족 예측

클래스 레이블 명은 `TARGET` 이며 1이면 불만을 가진 고객, 0이면 만족한 고객이다.

데이터 전처리

- 피처가 371개 존재한다.
- 111개의 피처가 float형, 260개의 피처가 int 형으로 모든 피처가 숫자 형이며, Null 값은 없다.

  → 불만족인 고객은 4%이다.
- ID 값 drop, var3 칼럼의 -999999값을 2로 대체한다.

XGBoost 모델 학습과 하이퍼 파라미터 튜닝

1. XGBoost 모델 생성 후 예측 결과를 ROC AUC로 평가하자.
2. XGBoost의 하이퍼 파라미터 튜닝 수행(8개의 하이퍼 파라미터 경우의 수)
3. 최적의 하이퍼 파라미터 찾고, estimator와 early_stopping_rounds를 높게 설정하고 수행

→ **수행 시간이 상당히 오래 걸린다.**

LightGBM 모델 학습과 하이퍼 파라미터 튜닝

1. LightGBM을 통해 측정 시간이 XGBoost보다 감소했음을 알 수 있다.

09 분류 실습 - 캐글 신용카드 사기 검출

Class는 0이면 정상적인 신용카드 트랜잭션 데이터, 1이면 신용카드 사기 트랜젝션을 의미한다.

전체 데이터의 0.172%만이 레이블 값이 1, 즉 사기 트랜잭션이다.

언더 샘플링과 오버 샘플링의 이해

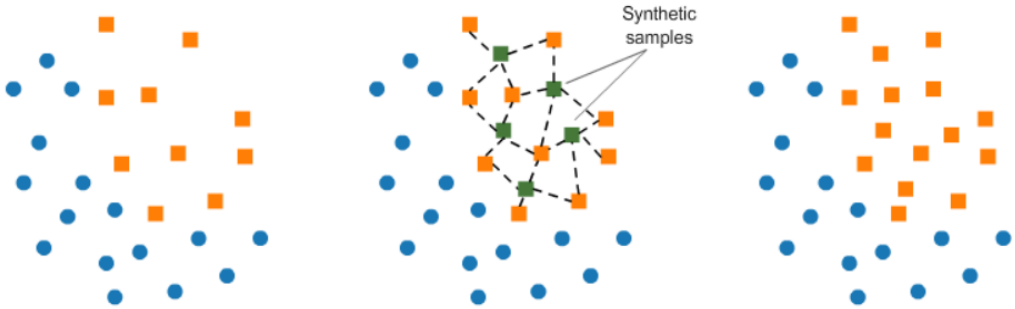
예측 성능의 문제가 발생하는 이유

이상 레이블을 가지는 데이터 건수가 정상 레이블을 가진 데이터 건수에 비해 너무 적을 때 발생함

→ 극도로 불균형한 레이블 값 분포로 인한 문제점을 해결하기 위해선 **적절한 학습 데이터를 확보해야한다.**

대표적으로 **오버 샘플링**과 **언더 샘플링** 방법이 있다.

- **언더 샘플링** : 많은 레이블을 가진 데이터 세트를 적은 레이블을 가진 데이터 세트 수준으로 감소
- **오버 샘플링** : 적은 레이블을 가진 데이터 세트를 많은 레이블을 가진 데이터 세트 수준으로 감소



SMOTE(Synthetic Minority Over-sampling Technique) 방법

- **적은 데이터 세트에 있는 개별 데이터들의 K 최근접 이웃을 찾아서 이 데이터와 K개 이웃들의 차이를 일정 값으로 만들어서 기존 데이터와 약간 차이가 나는 새로운 데이터들을 생성하는 방식**

데이터 일차 가공 및 모델 학습/예측/평가

1. 신용카드 사기 검출 모델 생성
2. 데이터 사전 가공(Time 피처 삭제)

3. Stratified 방식으로 테스트 데이터 세트의 30%를 추출해 학습/테스트 데이터 세트의 레이블 값 분포도 동일하게 만들기

   → 학습 데이터 레이블과 테스트 데이터 레이블 값이 서로 0.172%, 0.173%로 분할됨을 확인할 수 있다.

4. 로지스틱 회귀와 LightGBM모델로 데이터 가공 수행

   - **로지스틱 회귀**

```
오차행렬
[[85269    26]
 [   58    90]]
정확도: 0.9990, 정밀도: 0.7759, 재현율: 0.6081, F1: 0.6818, AUC:0.8039
```

   - **LightGBM**

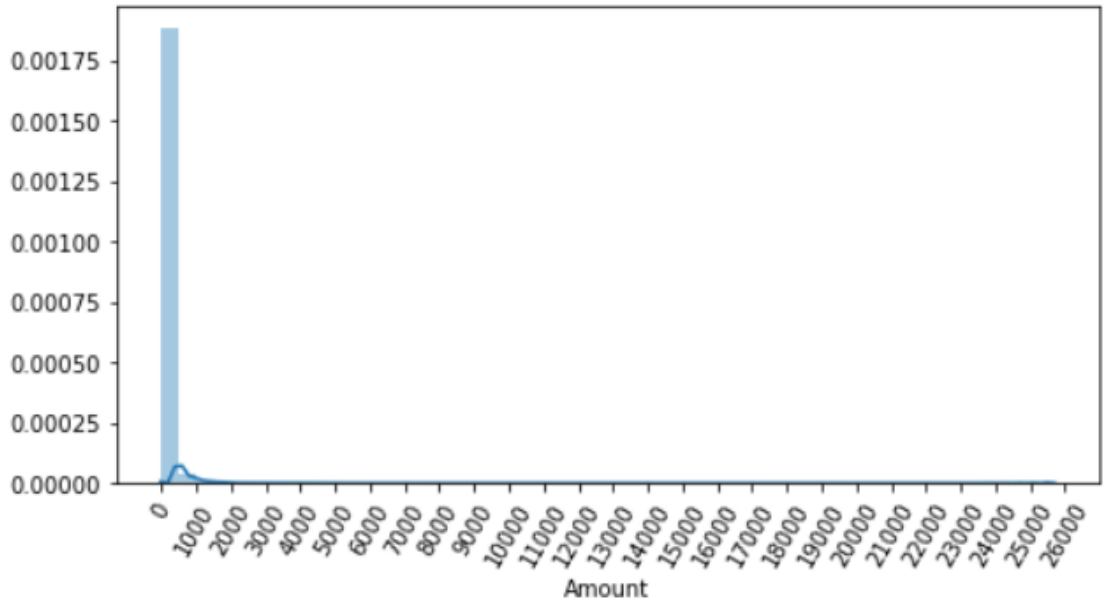
```
오차행렬
[[85290     5]
 [   36   112]]
정확도: 0.9995, 정밀도: 0.9573, 재현율: 0.7568, F1: 0.8453, AUC:0.8783
```

   → `get_model_train_eval()` 인자로 Estimator 객체와 학습/테스트 데이터 세트를 입력받아 학습, 평가 수행

   → `LGBMClassifier` 호출 후, 각 데이터 세트 피팅 후 평가

   결과 : 로지스틱 회귀보다 재현율과 AUC가 높음을 알 수 있다.

데이터 분포도 변환 후 모델 학습/예측/평가



Amount(카드 사용금액)이 1000불 이하인 데이터가 대부분이다.

→ Amount를 표준 정규 분포 형태로 변환한 뒤에 로지스틱 회귀의 예측 성능을 측정하자.

1. StandardScaler

`StandardScaler` 를 이용해 Amount 피처를 정규 분포 형태로 변환하는 코드로 변경한다.

1. `StandardScaler` 를 통해 변환된 Amount를 Amount_Scaled로 피처명 변경 후 앞 칼럼으로 입력
2. 기존 Time, Amount 피처 삭제
3. Amount를 정규 분포 형태로 변환후 로지스틱 회귀 및 LightGBM 수행
4. **결과**

```
### 로지스틱 회귀 예측 성능 ###
오차행렬
[[85281    14]
 [   58    90]]
정확도: 0.9992, 정밀도: 0.8654, 재현율: 0.6081, F1: 0.7143, AUC:0.8040

### LightGBM 예측 성능 ###
오차행렬
[[85290     5]
 [   37   111]]
정확도: 0.9995, 정밀도: 0.9569, 재현율: 0.7500, F1: 0.8409, AUC:0.8750
```

→ **성능이 크게 개선되지 않았다.**

2. Log 변환

log 변환은 **데이터 분포도가 심하게 왜곡되어 있을 경우 적용하는 중요 기법** 중에 하나이다.

1. 넘파이의 `log1p()` 함수를 이용해 Amount를 로그 변환하자.
2. Amount 피처를 로그 변환 후 다시 로지스틱 회귀와 LightGBM 모델 적용
3. **결과**

```
### 로지스틱 회귀 예측 성능 ###
오차행렬
[[85283    12]
 [   59    89]]
정확도: 0.9992, 정밀도: 0.8812, 재현율: 0.6014, F1: 0.7149, AUC:0.8006

### LightGBM 예측 성능 ###
오차행렬
[[85290     5]
 [   35   113]]
정확도: 0.9995, 정밀도: 0.9576, 재현율: 0.7635, F1: 0.8496, AUC:0.8817
```

→ **약간 성능이 개선됨을 확인할 수 있다.**

이상치 데이터 제거 후 모델 학습/예측/평가

이상치 데이터(Outlier)는 전체 데이터의 패턴에서 벗어난 이상 값을 가진 데이터이다.

이상치를 찾아내어 제거하고 모델을 평가한다.

이상치를 찾는 방법

- **IQR(Inter Quantile Range) 방식**을 적용한다.

  → 사분위 값의 편차를 이용하는 기법, Box Plot 방식으로 시각화가 가능하다.

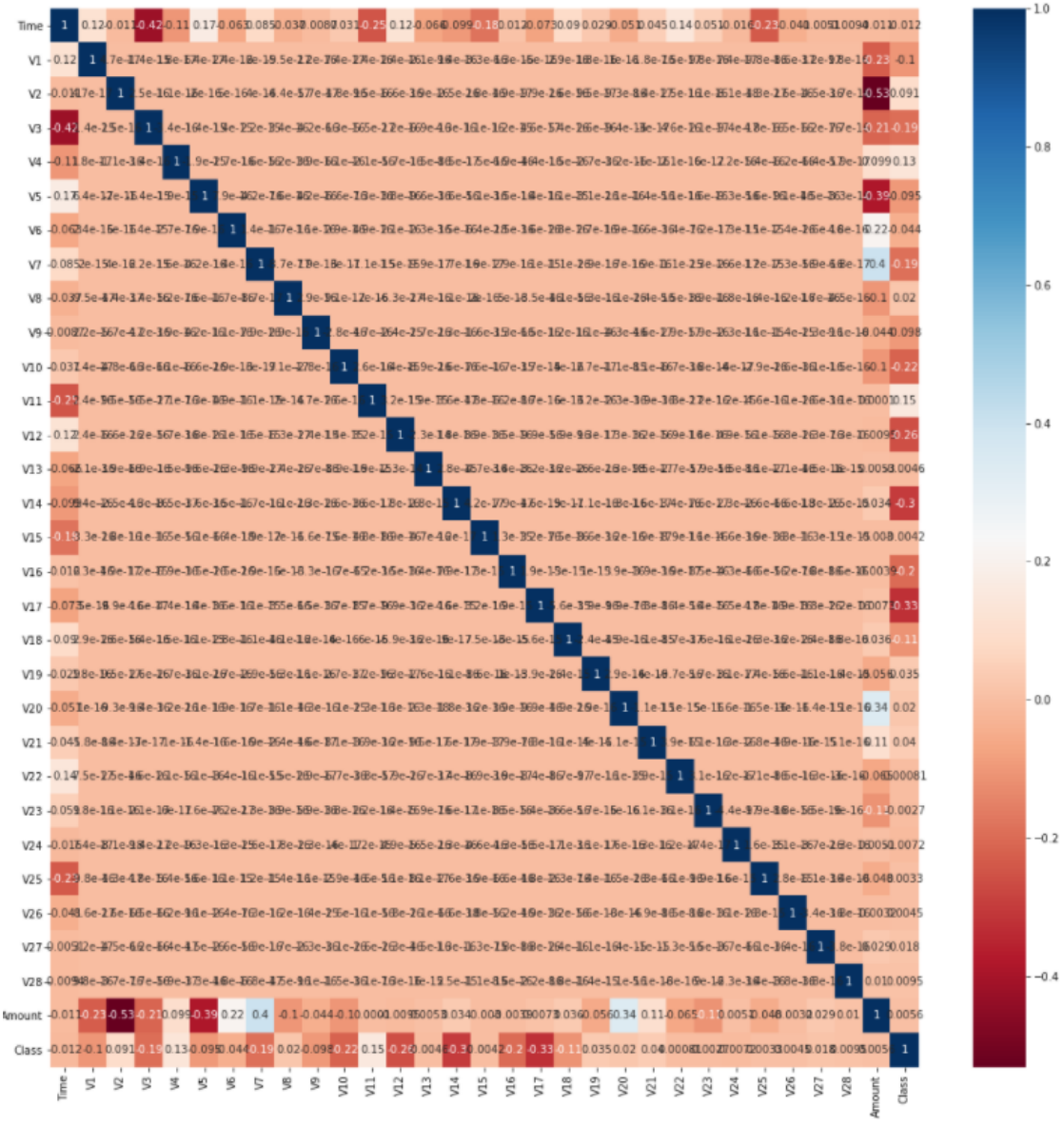
사분위란?

전체 데이터를 값이 높은 순으로 정렬하고 이를 1/4씩 구간을 분할하는 것

이 때, **25%~75%의 범위의 데이터**를 **IQR**이라고 한다.

IQR을 이용해 이상치를 검출하는 방식

이상치 : **25%의 범위에 1.5*IQR값을 뺀 지점** 이하, **75% 범위에 1.5*IQR값을 더한 지점** 이상



Class와 음의 상관관계가 높은 V14와 17 중 14에 대해서만 이상치를 찾아 제거해보자.

1. `get_outlier()` 함수를 통해 DataFrame과 이상치를 검출한 칼럼을 입력받는다.
2. `percentile()` 을 이용해 1/4와 3/4를 구하고 이에 기반에 IQR을 계산한다.
3. 최솟값보다 작거나 최댓값도가 큰 값을 이상치로 설정하고 해당 이상치가 있는 DataFrame Index로 반환한다.
4. **결과**

```
이상치 데이터 인덱스: Int64Index([8296, 8615, 9035, 9252], dtype='int64')
```

5. 이상치를 삭제하는 로직을 `get_processed_df()` 에 추가해 데이터를 가공한 뒤 로지스틱 회귀와 LightGBM 모델에 다시 적용하자.

6. **결과**

```
### 로지스틱 회귀 예측 성능 ###
오차행렬
[[85281    14]
 [   48    98]]
정확도: 0.9993, 정밀도: 0.8750, 재현율: 0.6712, F1: 0.7597, AUC:0.8355

### LightGBM 예측 성능 ###
오차행렬
[[85290     5]
 [   25   121]]
정확도: 0.9996, 정밀도: 0.9603, 재현율: 0.8288, F1: 0.8897, AUC:0.9144
```

→ 둘 다 예측 성능이 **크게 향상**된 것을 확인할 수 있다.

SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가

반드시 **학습 데이터 세트만 오버 샘플링**을 해야 한다.

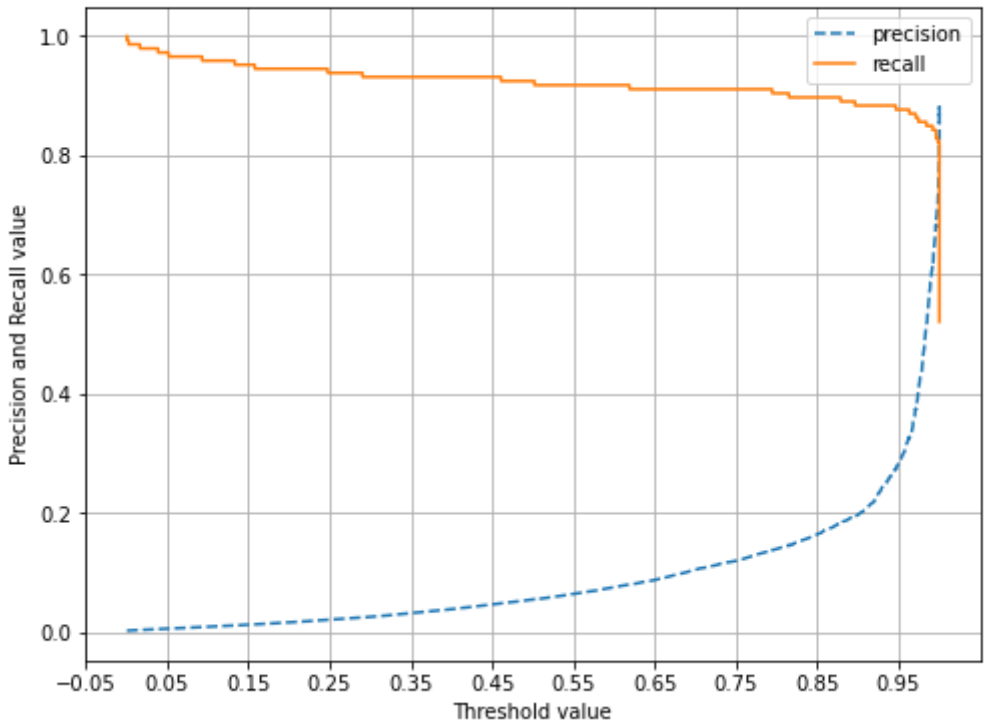
```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ((199362, 29), (199362,))
SMOTE 적용 후 학습용 피처/레이블 데이터 세트: ((398040, 29), (398040,))
SMOTE 적용 후 레이블 값 분포:
1    199020
0    199020
Name: Class, dtype: int64
```

SMOTE 적용 후 **2배 가까운 데이터 증식**을 확인할 수 있다.

로지스틱 회귀 모델 적용

```
오차행렬
[[82937  2358]
 [   11   135]]
정확도: 0.9723, 정밀도: 0.0542, 재현율: 0.9247, F1: 0.1023, AUC:0.9485
```

정밀도가 떨어진 이유 : 너무 많은 Class = 1을 학습하면서 실제 데이터 세트에서 예측을 지나치게 Class = 1로 적용해 정밀도가 급격히 떨어지게 되었다.



→ 올바른 예측 모델을 생성하지 못했다.

LightGBM 적용

```
오차행렬
[[85283    12]
 [   22   124]]
정확도: 0.9996, 정밀도: 0.9118, 재현율: 0.8493, F1: 0.8794, AUC:0.9246
```

재현율은 높지만 정밀도는 낮아졌다.

→ SMOTE를 적용하면 Class = 1의 데이터가 많아지므로, 예측의 결과의 차이로 인해 정밀도가 떨어질 수 있다. 그러나, **좋은 SMOTE 패키지일 수록, 재현율 증가율은 높이고, 정밀도 감소율은 낮춘다.**

10 스태킹 앙상블

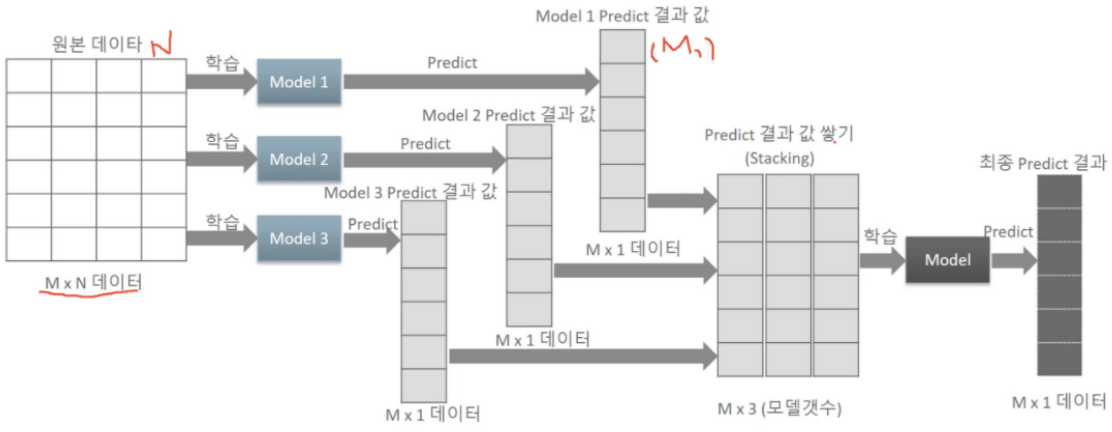
스태킹 : 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측을 수행함.

스태킹의 필요 모델

1. 개별적인 기반 모델
2. 이 개별 기반 모델의 예측 데이터를 학습 데이터로 만들어서 학습하는 **최종 메타 모델**

스태킹 모델의 핵심

- 여러 개별 모델의 예측 데이터를 각각 스태킹 형태로 결합해 최종 메타 모델의 학습/테스트 용 피처 데이터 세트를 만드는 것.



Stacking의 구체적인 동작 방식

CV 세트 기반의 스태킹

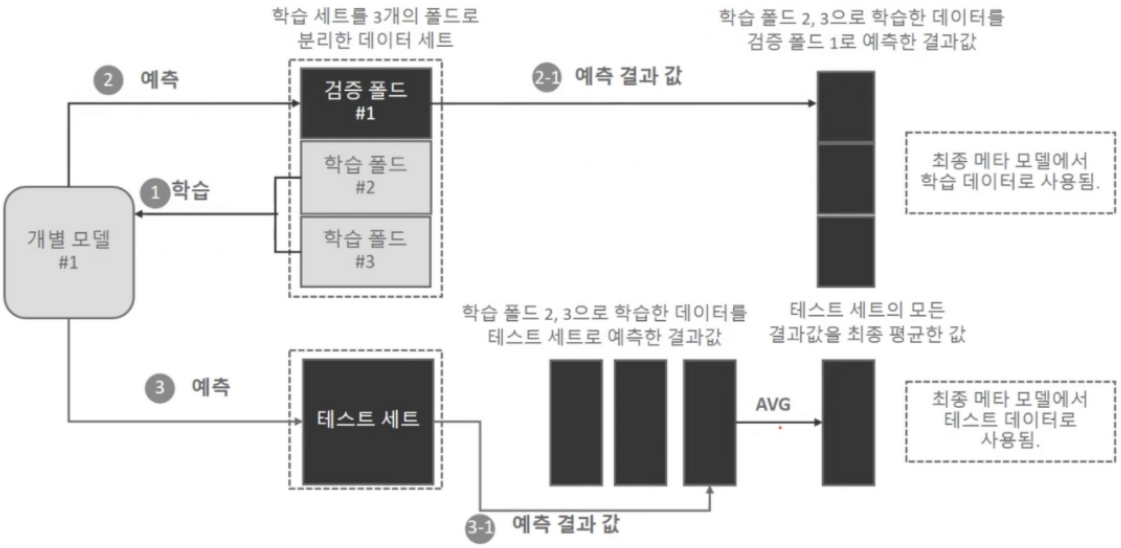
스태킹은 메타 모델인 로지스틱 회귀 모델 기반에서 최종 학습할 때 테스트용 레이블 데이터 세트를 기반으로 학습했기 때문에 **과적합**이 발생할 수 있다.

CV 세트 기반의 스태킹은 개선을 위해 개별 모델들이 각각 교차 검증으로 메타 모델을 위한 **학습용/테스트용 스태킹 데이터**를 생성한 뒤에 이를 기반으로 메타 모델이 학습과 예측을 수행한다.

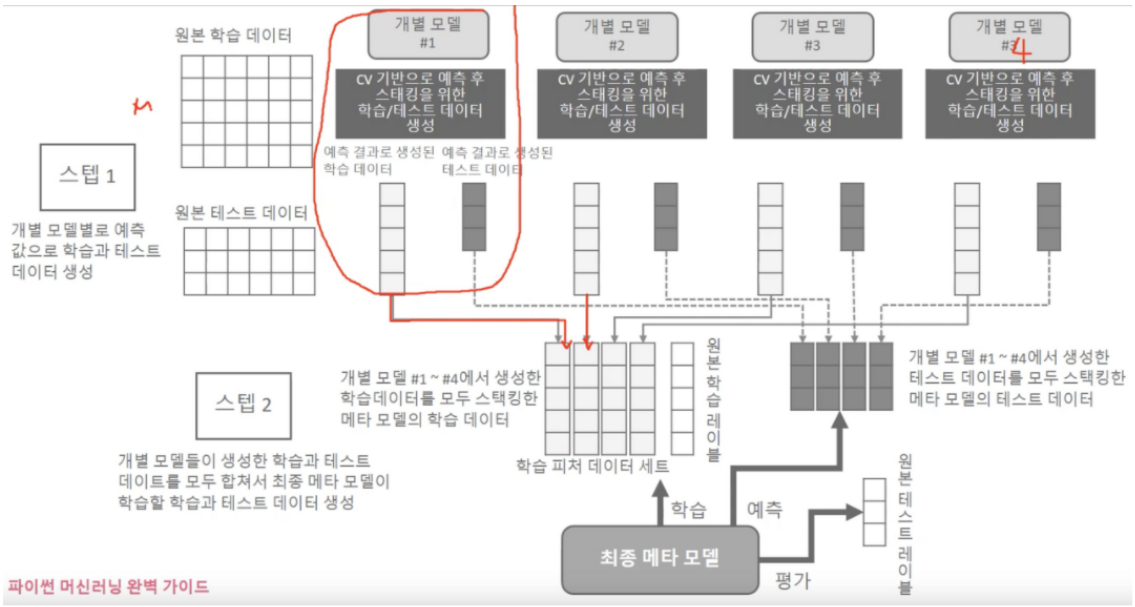
1. 각 모델별로 원본 학습/테스트 데이터를 예측한 결과 값을 기반으로 메타 모델 생성
2. 개별 모델들이 생성한 학습용 데이터와 테스트용 데이터를 각각 따로 스태킹으로 합쳐서 진행
3. 메타 모델은 최종적으로 생성된 학습 데이터와 원본 학습 테스트 데이터의 레이블 데이터를 학습한뒤, 최종적으로 생성된 테스트 데이터 세트를 예측하고, 원본 테스트 데이터의 레이블 데이터를 기반으로 평가한다.

교차 검증을 통해 스텝별로 수행하는 로직을 자세히 살펴보자

1. 학습용 데이터를 3개의 폴드로 나눈 후 2개는 학습을 위한 데이터 폴드로, 나머지 1개는 검증을 위한 데이터 폴드로 나눈다. 학습데이터를 개별 모델을 학습시킨다.
2. 이렇게 학습된 개별 모델은 검증 폴드 1개를 예측하고 그 결과를 저장한다. 이러한 로직을 3번 반복하면서, 학습 데이터와 검증데이터를 변경하면서 학습 후 예측결과를 별도로 저장한다.
3. 2개의 학습 폴드 데이터로 학습된 개별 모델은 원본 테스트 데이터를 예측하여 예측값을 생성한다.
4. 이러한 로직을 3번 반복하면서, 이 예측값의 평균으로 최종 결과값을 생성하고, 이를 메타 모델을 위한 테스트 데이터로 사용한다.



개별 모델 내부에서 CV Stacking이 동작하는 과정



개별모델이 도출한 결과값들을 기반으로 Stacking하는 과정