분류(Classification)

: 학습 데이터로 주어진 데이터의 feature와 label 값(결정 값, 클래스 값)을 머신 러닝 알고리즘으로 학습해 모델을 생성하고, 이렇게 생성된 모델에 새로운 데이터 값이 주어졌을 때, 미지의 레이블 값을 예측하는 것

- Bayes 통계와 생성 모델에 기반한 Naïve Bayes
- 독립변수와 종속변수의 선형 관계성에 기반한 Logistic Regression(회귀지만 Sigmoid 이용하여 분류(Classification)도 가능)
- 데이터 균일도에 따른 규칙 기반의 결정 트리(Decision Tree)
- 개별 클래스 간의 최대 분류 마진을 효과적으로 찾아주는 SVM(Support Vector Machine)
- 근접 거리를 기준으로 하는 최소 근접(Nearest Neighbor) 알고리즘
- 심층 연결 기반의 신경망(Deep Neural Network)
- 서로 다른 (또는 같은) 머신러닝 알고리즘을 결합한 **앙상블(Ensemble)**

결정 트리(Decision Tree)

- 쉽고 유연하게 적용 가능한 알고리즘이며, 스케일링이나 정규화 등의 사전 가공의 영향이 매우 적다. 하지만, 예측 성능을 향상시키기 위해 복잡한 규칙 구조를 가져야 하며, 이로 인한 과적합(Overfitting)이 발생해 반대로 예측 성능이 저하될 수 있다.
- 하지만 이런 단점이 앙상블 기법에서는 오히려 장점으로 작용하는데, 앙상블은 매우 많은 여러 개의 약한 학습기를 결합해 확률적 보완과 오류가 발생한 부분에 대한 가중치를 계속 업데이트하면서 예측 성능을 향상시키는데, 결정 트리가 좋은 **약한 학습기**(Weak Learner)가 되기 때문이다.

새로운 규칙 조건마다 subtree(branch)를 만든다. 어떻게 처음에 답에 근접한 질문을 던지는 지가 중요.

➔ 스무고개와 비슷

초기에 이런 질문을 하는 것이 알고리즘의 성능을 크게 좌우한다.

- 학습을 통해 규칙을 자동으로 찾아내 트리 기반의 분류 기준을 만듦(if-else 기반 규칙)
- 데이터의 어떤 기준을 바탕으로 할 것인지가 성능을 크게 좌우한다.

데이터의 균일도.

➔ 반, 반에 최대한 가까운 값으로 분할할 수 있어야 한다.(처음에 가까울수록)

정보 균일도 측정 방법

- 정보 이득(Information Gain) : 정보 이득은 엔트로피라는 개념을 기반으로 한다. **서로 다른 값이 섞여 있으면 엔트로피가 높고**, **같은 값이 섞여 있으면 엔트로피가 낮다**. 정보 이득 지수는 1에서 엔트로피 지수를 뺀 값이다. (1-엔트로피) 결정 트리는 정보 이득이 높은 속성을 기준으로 분할한다.

정보 이득 지수 : (1-엔트로피)이므로, 다양성이 낮은 수록(같은 값이 섞여 있을수록) 엔트로피가 낮아, 정보 이득이 높다.

- 지니 계수 : 0이 가장 평등, 1이 가장 불평등한 지수. 다양한 값을 가질수록 평등하며, 특정 값으로 쏠릴 경우에는 불평등한 값이 된다.

지니 계수 : 다양성이 낮을수록(불평등 할수록) 균일도가 높다.

: 둘 다 상식적으로 최종적으로 하나의 종류만 남아 있을 때가 균일도가 높다

분류로 나누며 -> 모든 데이터 집합이 결정될 때까지 반복.

max_depth

- default : None -> 끝까지.(깊이)
- **min_samples_split** 보다 작아질 때까지 깊이를 증가시킨다.

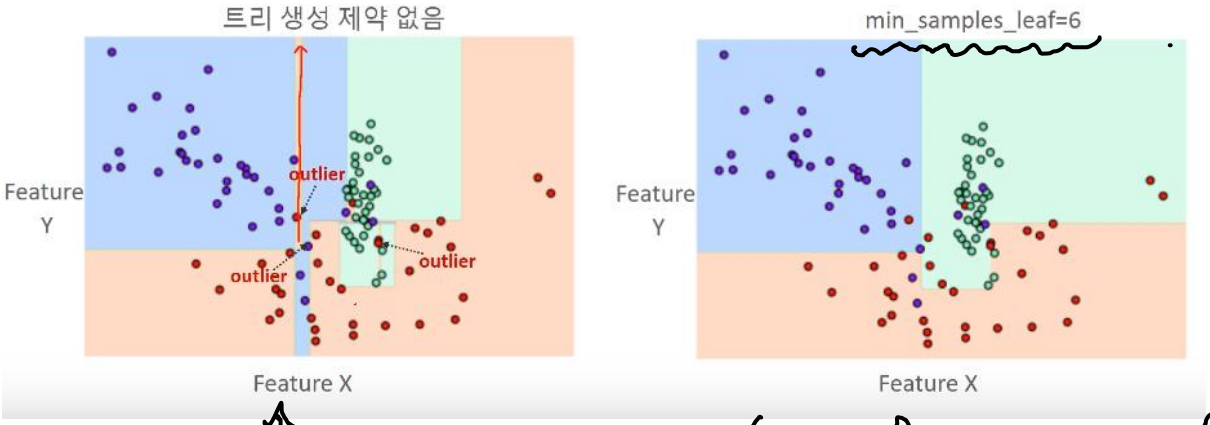
max_features

: 최적의 분할을 위해 고려할 최대 feature 수

Graphviz 다운로드 및 이용

- petal length
- gini 지니계수
- samples 데이터 건수
- value = [] 0 : Setosa, 1 : Versicolor, 2 : Virginica 각각의 개수를 나타냄. value = [41, 40, 39]라면 Setosa가 41개, Versicolor가 40개, Virginica : 39라는 뜻이다.
- 색깔이 진할수록 단일화 분류 값을 가지고 있다. 연하면 불확실성이 높다

feature_importances_ 을 통해 중요한 Feature들을 선택할 수 있게 정보를 제공한다.



overfitting 예시.

max_depth에 따른 정확도 값을 확인할 수 있다. train에서는 max_depth가 클수록 정확도가 높았으나, test set 에서는 오히려 높을수록 정확도가 낮은 경우가 있는 등 일관성이 없고, max_depth가 정확성을 보장하지 못한다는 것을 알 수 있었다.

앙상블 학습

: 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법

앙상블의 유형

- 일반적으로는 보팅(Voting), 배깅(Bagging), 부스팅(Boosting)으로 구분할 수 있으며, 이외에 스태킹(Stacking) 등의 기법이 있다.
- **대표적인 배깅은 랜덤 포레스트(Random Forest) 알고리즘이 있으며, 부스팅은 에이다 부스팅, 그래디언트 부스팅, XGBoost, LightGBM** 등이 있다. 정형 데이터의 분류나 회귀에서는 GBM 부스팅 계열의 앙상블이 전반적으로 높은 예측 성능을 나타낸다.
- 넓은 의미로는 서로 다른 모델을 결합한 것을 앙상블로 지칭하기도 한다.

앙상블의 특징

- 단일 모델의 약점을 다수의 모델을 결합하여 보완
- 뛰어난 성능을 가진 모델들로만 구성하는 것보다 성능이 떨어지더라도 서로 다른 유형의 모델을 섞는 것이 오히려 전체 성능에 도움이 될 수 있다.
- 랜덤 포레스트 및 뛰어난 부스팅 알고리즘은 모두 결정 트리 알고리즘을 기반 알고리즘으로 적용한다. (결정 트리의 단점인 Overfitting을 수십~수천 개의 많은 분류기를 결합해 보완하고 장점인 직관적인 분류기준은 강화하는 방식으로 이용한다.)

보팅과 배깅

- 보팅과 배깅은 여러 개의 분류기가 투표(Vote)를 통해 최종 예측 결과를 결정하는 방식
- 보팅은 일반적으로 **서로 다른 알고리즘**을 가진 분류기를 결합하는 것이다. (분류기들이 모두 같은 Data Set 공유)
- 배깅은 각각의 분류기가 **모두 같은 유형의 알고리즘** 기반이지만, **데이터 샘플링을 서로 다르게** 가져가면서 학습을 수행하여 보팅을 수행하는 것이다. (각각의 분류기가 서로 다른 Sampling Data 사용)

보팅 유형

- 하드 보팅(Hard Voting) : 다수결이라고 생각하면 된다.
- 소프트 보팅(Soft Voting) : Classifier 들의 class 별 확률을 평균하여 결정

일반적으로 하드 보팅보다는 소프트 보팅이 예측 성능이 상대적으로 우수하여 주로 사용된다.

사이킷런은 VotingClassifier 클래스를 통해 보팅(Voting)을 지원

배깅 – Random Forest

- 랜덤 포레스트는 다재 다능한 알고리즘이다. 빠른 수행 속도, 높은 예측 성능을 보인다.
- 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 최종적으로 모든 분류기가 보팅을 통해 예측 결정을 한다.
- 여러 개의 데이터 세트를 중첩되게 분리하는 것을 부트스트래핑(bootstrapping) 분할 방식이라고 한다. (원본 데이터의 건수가 10개인 학습 데이터 세트에 랜덤 포레스트를 3개의 결정 트리 기반으로 학습하려고 n_estimators = 3으로 하이퍼 파라미터를 부여하면 다음과 같이 데이터 서브 세트가 만들어진다.)

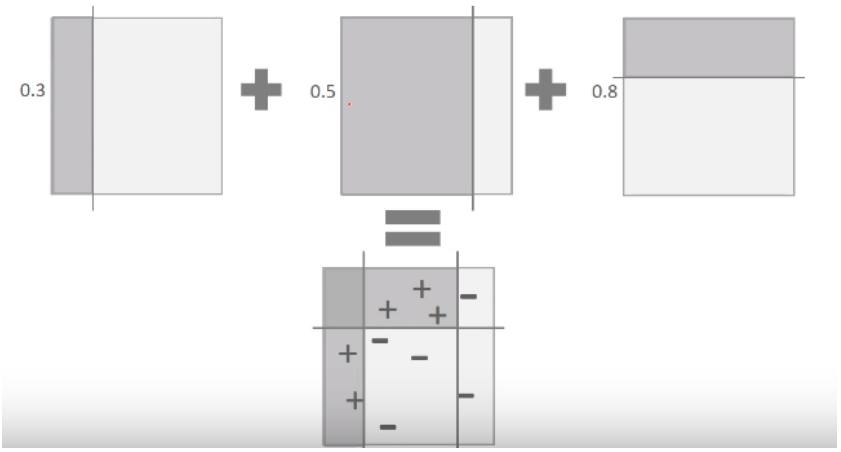
RandomForestClassifier HyperParameter

- n_estimators : 결정 트리의 개수를 지정. default는 10개.
- max_features : 결정 트리에서 사용된 max_features 파라미터와 같이 최적의 분할을 위해 고려할 최대 feature 수. 단 default가 None이 아니라 'auto'. 즉, 'sqrt'와 같다. 따라서 랜덤 포레스트의 트리를 분할하는 feature를 참조할 때 전체 피처가 아니라 전체 피처의 루트만큼을 참조한다.
- max_depth나 min_samples_leaf와 같이 결정 트리에서 과적합을 개선하기 위해 사용되는 파라미터가 랜덤 포레스트에서 동일하게 적용된다.

부스팅(Boosting)

- 부스팅 알고리즘은 **여러 개의 약한 학습기(Weak Learner)를 순차적(수행 시간이 오래 걸림)으로 학습-예측**하면서 **잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선**해 나가면서 학습하는 방식이다.
- 부스팅의 대표적인 구현은 AdaBoost(Adaptive boosting)과 Gradient Boosting 이 있다.

AdaBoost 예



GBM

GBM도 AdaBoost와 유사하나, 가중치 업데이트를 순차적으로 Gradient Descent를 이용하여 계속 수행하는 점이 큰 차이다.

GradientBoostingClassifier Hyperparameter

- loss
- learning_rate

- n_estimators : weak learner의 개수.(default = 100) weak learner 가 순차적으로 오류를 보정하므로 개수가 많을수록 예측 성능이 좋아질 수 있으나, 개수가 많을수록 수행 시간이 오래 걸린다. (수행 시간이 오래 걸리는 편)
- subsample : weak learner가 학습에 사용하는 데이터 샘플링 비율(default는 1이며, 이는 전체 학습 데이터를 기반으로 학습한다는 의미) **overfitting 이 염려되면 작은 값으로 설정 필요**

XGBoost

- 뛰어난 성능
- GBM 대비 빠른 수행 시간(CPU 병렬 처리, GPU 지원)
- 다양한 성능 향상 기능
  - 규제 (Regularization) 기능 탑재
  - Tree Pruning(Node들을 다 만들었다가 leaf node부터 다시 검증하여 pruning)

    : leaf node가 overfitting 을 반영한다고 보고 penalty를 부여하는 등의 방식을 통해 가지치기
- 다양한 편의 기능
  - EarlyStopping : 수행 속도 감소 및 overfitting 방지
  - 내장된 교차 검증
  - 결측값 자체 처리

사이킷런 제공

분류에서는 XGBClassifier

회귀에서는 XGBRegressor 사용

항목	파이썬 Wrapper	사이킷런 Wrapper
사용 모듈	from xgboost as xgb	from xgboost import XGBClassifier
학습용과 테스트용 데이터 세트	DMatrix 객체를 별도 생성 train = xgb.DMatrix(data=X_train , label=y_train) DMatrix 생성자로 피처 데이터 세트와 레이블 데이터 세트를 입력	넘파이나 판다스를 이용
학습 API	Xgb_model=xgb.train( ) Xgb_model은 학습된 객체를 반환 받음	XGBClassifer.fit( )
예측 API	xgb.train( )으로 학습된 객체에서 predict( ) 호출. 즉 Xgb_model.predict( ) 이때 반환 결과는 예측 결과가 아니라 예측 결과를 추정하는 확률값 반환	XGBClassifer.predict( ) 예측 결과값 반환
피처 중요도 시각화	plot_importance( ) 함수 이용	plot_importance( ) 함수 이용

파이썬 Wrapper	사이킷런 Wrapper	하이퍼 파라미터 설명
eta	learning_rate	GBM의 학습률(learning rate)과 같은 파라미터입니다. 0에서 1 사이의 값을 지정하며 부스팅 스텝을 반복적으로 수행할 때 업데이트되는 학습률 값. 파이썬 래퍼 기반의 xgboost를 이용할 경우 디폴트는 0.3. 사이킷런 래퍼 클래스를 이용할 경우 eta는 learning_rate 파라미터로 대체되며, 디폴트는 0.1입니다.
num_boost_rounds	n_estimators	사이킷런 앙상블의 n_estimators와 동일. 약한 학습기의 개수(반복 수행 회수)
min_child_weight	min_child_weight	결정트리의 min_child_leaf와 유사. 과적합 조절용
max_depth	max_depth	결정트리의 max_depth와 동일. 트리의 최대 깊이
sub_sample	subsample	GBM의 subsample과 동일. 트리가 커져서 과적합되는 것을 제어하기 위해 데이터를 샘플링하는 비율을 지정합니다. sub_sample=0.5로 지정하면 전체 데이터의 절반을 트리를 생성하는 데 사용합니다. 0에서 1사이의 값이 가능하나 일반적으로 0.5 ~ 1 사이의 값을 사용합니다.

사이킷런 Wrapper의 경우 GBM에 동일한 하이퍼 파라미터가 있다면 이를 사용하고 그렇지 않다면 파이썬 Wrappe의 하이퍼 파라미터를 사용

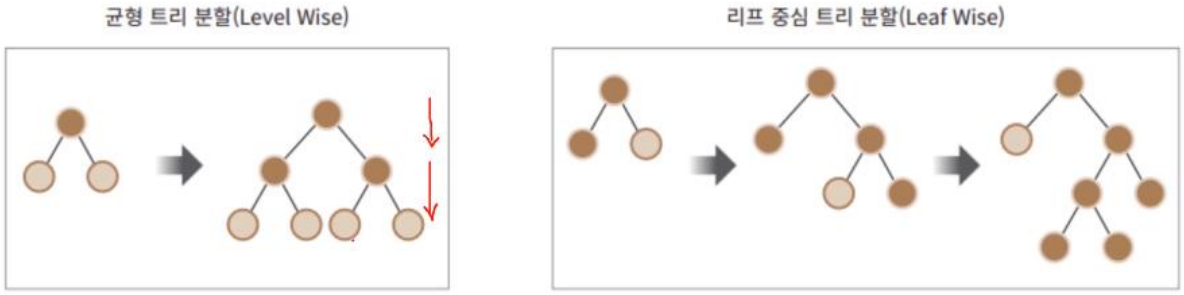
파이썬 Wrapper	사이킷런 Wrapper	하이퍼 파라미터 설명
lambda	reg_lambda	L2 규제(Regularization) 적용 값. 기본값은 1임. 값이 클 수록 규제 값이 커짐. 과적합 제어
alpha	reg_alpha	L1 규제(Regularization) 적용 값. 기본값은 0임. 값이 클 수록 규제 값이 커짐. 과적합 제어
colsample_bytree	colsample_bytree	GBM의 max features와 유사합니다. 트리 생성에 필요한 피처(컬럼)을 임의로 샘플링 하는 데 사용됩니다. 매우 많은 피처가 있는 경우 과적합을 조정하는 데 적용합니다
scale_pos_weight	scale_pos_weight	특정 값으로 치우친 비대칭한 클래스로 구성된 데이터 세트의 균형을 유지하기 위한 파라미터입니다. 기본값은 1
gamma	gamma	트리의 리프 노드를 추가적으로 나눌지를 결정할 최소 손실 감소 값입니다. 해당 값보다 큰 손실(loss)이 감소된 경우에 리프 노드를 분리합니다. 값이 클수록 과적합 감소 효과가 있습니다

LightGBM

: XGBoost 이후에 나와 XGBoost의 단점을 보완하고 성능을 높이는데 초점이 맞춰져 있다.

XGBoost 대비 장점

- 더 빠른 학습과 예측 수행 시간(수행 시간은 설왕설래가 있다.)
- 더 작은 메모리 사용량
- 카테고리형 feature의 자동 변환과 최적 변환. One-hot Encoding을 사용하지 않고도 카테고리형 feature를 최적으로 변환하고 이에 따른 노드 변환 수행



기존의 GBM은 한쪽 노드 밑으로 생겨서 그 노드 기준으로 계속 확장되고 overfitting 이 발생할 수 있다는 점 때문에 **균형 트리 분할(Level Wise)**을 사용하는데 반해,

LightGBM에서는 **리프 중심 트리 분할(Leaf Wise)**을 이용하는데, 좀 더 정확한 예측 성능을 보이며 오류를 줄여줄 수 있다고 판단.

유형	파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	
파라미터명	num_iterations	n_estimators	약한 학습기의 개수(반복 수행 회수)
	learning_rate	learning_rate	학습률(learning rate). 0에서 1 사이의 값을 지정하며 부스팅 스텝을 반복적으로 수행할 때 업데이트되는 학습률 값
	max_depth	max_depth	결정트리의 max_depth와 동일. 트리의 최대 깊이
	min_data_in_leaf	min_child_samples	리프 노드가 될 수 있는 최소 데이터 건수(Sample 수)
	bagging_fraction	subsample	트리가 커져서 과적합되는 것을 제어하기 위해 데이터를 샘플링하는 비율을 지정합니다. sub_sample=0.5로 지정하면 전체 데이터의 절반을 트리를 생성하는 데 사용합니다.
	feature_fraction	colsample_bytree	GBM의 max_features와 유사합니다. 트리 생성에 필요한 피처(컬럼)을 임의로 샘플링 하는 데 사용됩니다. 매우 많은 피처가 있는 경우 과적합을 조정하는 데 적용합니다

유형	파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	
파라미터명	lambda_l2	reg_lambda	L2 규제(Regularization) 적용 값. 기본값은 1임. 값이 클 수록 규제 값이 커짐. 과적합 제어
	lambda_l1	reg_alpha	L1 규제(Regularization) 적용 값. 기본값은 0임. 값이 클 수록 규제 값이 커짐. 과적합 제어
	early_stopping_round	early_stopping_rounds	학습 조기 종료를 위한 early stopping interval 값
	num_leaves	num_leaves	최대 리프노드 갯수
	min_sum_hessian_in_leaf	min_child_weight	결정트리의 min_child_leaf와 유사. 과적합 조절용

유형	파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	사이킷런 래퍼 XGBoost
파라미터명	num_iterations	n_estimators	n_estimators
	learning_rate	learning_rate	learning_rate
	max_depth	max_depth	max_depth
	min_data_in_leaf	min_child_samples	.N/A
	bagging_fraction	subsample	subsample
	feature_fraction	colsample_bytree	colsample_bytree
	lambda_l2	reg_lambda	reg_lambda
	lambda_l1	reg_alpha	reg_alpha
	early_stopping_round	early_stopping_rounds	early_stopping_rounds
	num_leaves	num_leaves	N/A
	min_sum_hessian_in_leaf	min_child_weight	min_child_weight

하드웨어가 충분하지 않은 경우에는 Feature Engineering을 먼저 수행한 뒤에 Hyperparameter 튜닝하는 것이 더 낫다. Hyperparameter 튜닝을 먼저 수행하는 경우 걸린 시간 대비 성능 향상이 기대한 만큼 이루어지지 않는 경우가 많다.

.head()

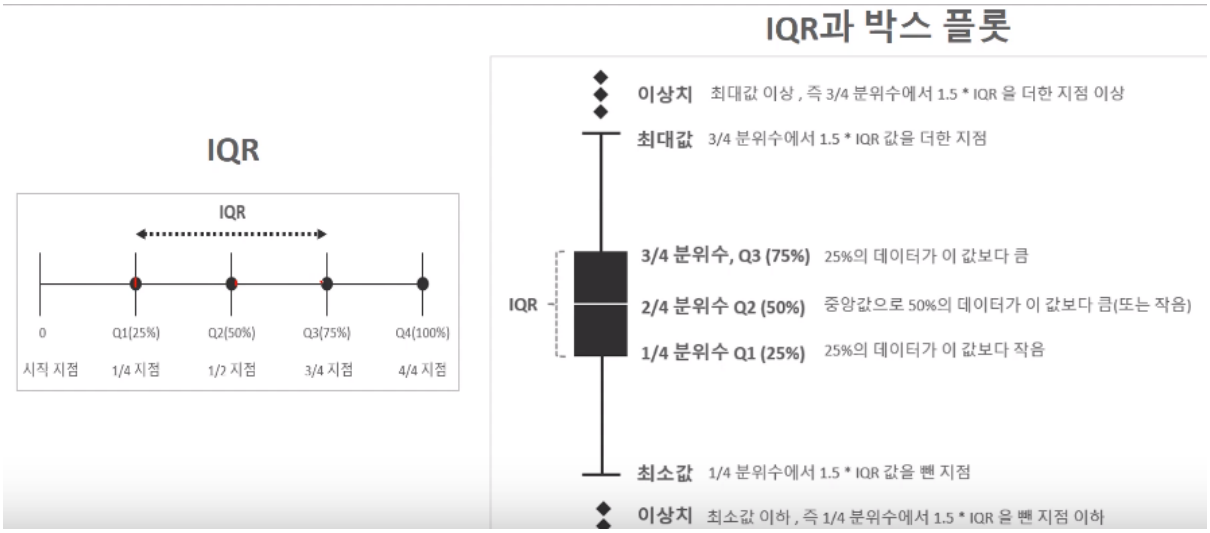
.info()

.describe()

.shape

로 데이터의 정보를 파악한 후 접근하는 것이 효율적이다.

IQR(Inter Quantile Range)를 이용한 Outlier Removal



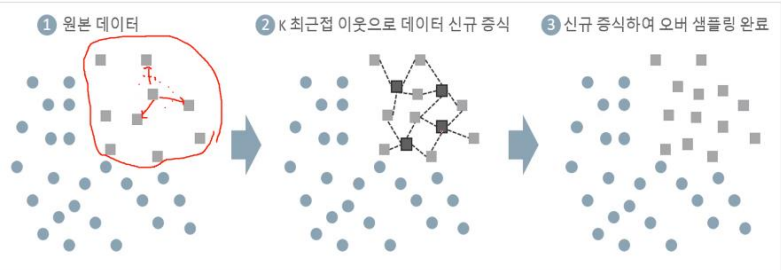
Undersampling 과 Oversampling

Undersampling : 많은 레이블을 가진 데이터 세트를 적은 레이블을 가진 데이터 세트 수준으로 감소 샘플링

Oversampling : 적은 레이블을 가진 데이터 세트를 많은 레이블을 가진 데이터 세트 수준으로 증가 샘플링

SMOTE(Synthetic Minority Over-Sampling Technique) 방식

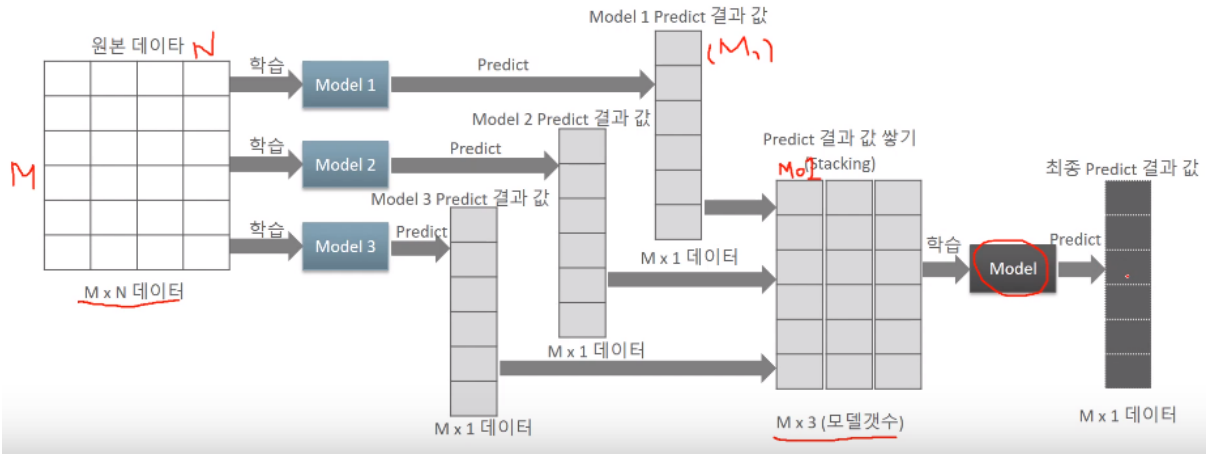
- 같은 레이블로 분류된 적은 데이터를 Oversampling 하기 위해서 사용하는 기법.
- K 최근접 이웃으로 사이에 있는 곳에 데이터를 신규 증식

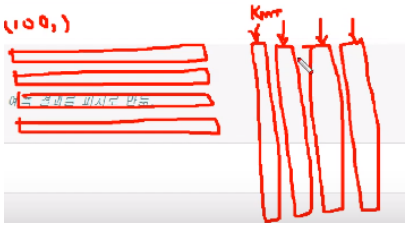


데이터 가공 유형	머신러닝 알고리즘	평가 지표		
		정밀도	재현율	ROC-AUC
데이터 가공 없음	로지스틱 회귀	0.8738	0.6081	0.9709
	LightGBM	0.9492	0.7568	0.9797
데이터 로그 변환	로지스틱 회귀	0.8824	0.6081	0.9721
	LightGBM	0.9576	0.7635	0.9786
이상치 데이터 제거	로지스틱 회귀	0.8829	0.6712	0.9747
	LightGBM	0.9680	0.8288	0.9831
SMOTE 오버 샘플링	로지스틱 회귀	0.0540	0.9247	0.9737
	LightGBM	0.9394	0.8493	0.9778

스태킹(쌓는다)

- **기반 모델들이 예측한 값을** Stacking 형태로 만들어서 메타 모델이 이를 학습하고 예측하는 모델





의 방식으로 스태킹 형태로 재생성

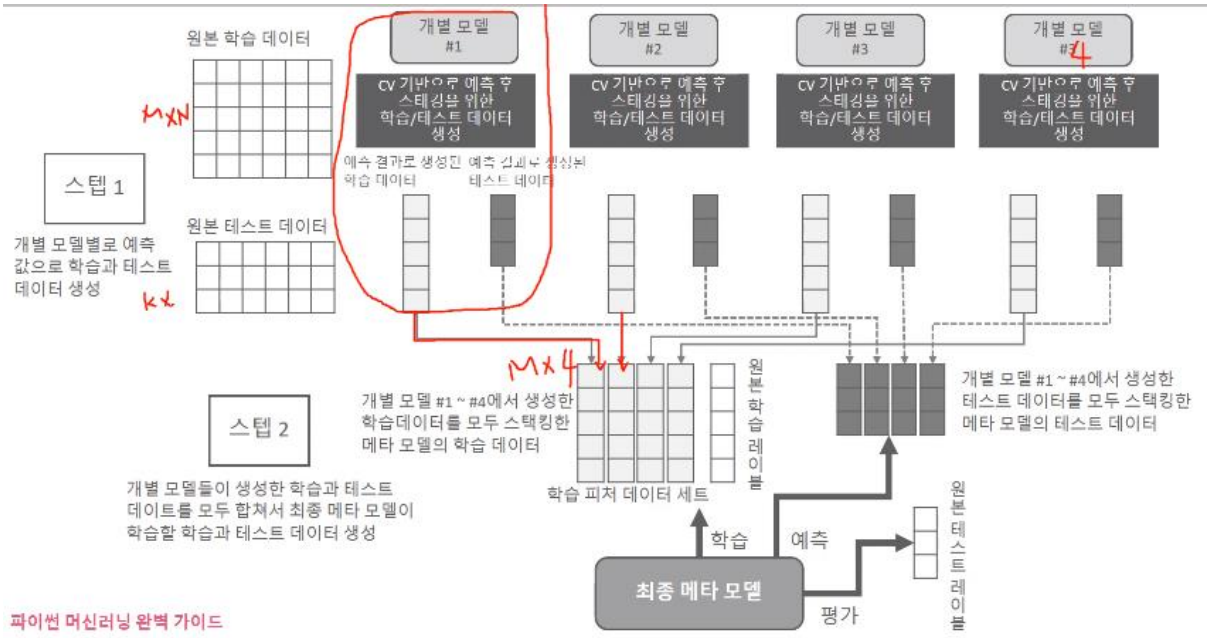
그냥 스태킹을 수행할 경우 overfitting의 발생이 잦다.

이런 overfitting 에 대한 개선을 위해 개별 모델들이 각각 교차 검증으로 메타 모델을 위한 학습용 스태킹 데이터 생성과 예측을 위한 테스트용 스태킹 데이터를 생성한 뒤 이를 기반으로 메타 모델이 학습과 예측을 수행한다. 이는 다음과 같이 2단계의 단계로 요약할 수 있다.

Step 1. **각 모델별로 원본 Train/Test 데이터를 예측한 결과값을 기반으로 메타 모델을 위한 Train/Test 데이터 생성 (매우 중요하고 어려움)**

Step 2.

- Step1에서 개별 모델들이 생성한 학습용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 학습할 **최종 Train dataset**을 생성한다. 마찬가지로 각 모델들이 생성한 테스트용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 예측할 **최종 Train dataset**을 생성한다.
- 메타 모델은 **최종 Train dataset와 원본 Train data의 레이블 데이터를 기반으로 학습**한 뒤, **최종 Test dataset를 예측하고, 원본 Test data의 레이블 데이터를 기반으로 평가**한다.



최종 정리

- 결정 트리와 결정 트리 기반의 앙상블
- 배깅과 부스팅
  - 랜덤 포레스트, GBM
- GBM의 기능을 더욱 향상 시킨 XGBoost, LightGBM
- 스태킹 모델

Feature Selection

모델을 구성하는 주요 feature들을 선택

- 불필요한 다수의 피처들로 인해 모델 성능을 떨어뜨릴 가능성 제거
- 설명 가능한 모델이 될 수 있도록 feature들을 선별
- Feature 값의 분포, NULL, feature 간의 상관관계, 결정 값과의 독립성 등을 고려
- 모델의 feature importance 기반

사이킷런

RFE(Recursive Feature Elimination)

- 모델 최초 학습 후 Feature 중요도 선정

- Feature 중요도가 낮은 속성들을 차례로 제거해 가면서 반복적으로 학습/평가를 수행하여 최적 feature 추출
- 수행 시간이 오래 걸리고, 낮은 속성들을 제거해 나가는 매커니즘이 정확한 Feature Selection을 찾는 목표에 부합하지 않을 수 있음

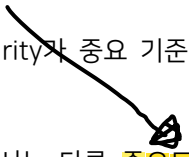
SelectionFromModel

- 모델 최초 학습 후 선정된 Feature 중요도에 따라 평균/중앙값의 특정 비율 이상인 Feature 들을 선택

Permutation(순열) importance

- 특정 Feature 들의 값을 완전히 변조했을 때(순열) 모델 성능이 얼마나 저하되는지를 기준으로 해당 feature의 중요도를 선정(중요한 것이라면 변조했을 때 모델 성능이 저하되어야 함.
- **학습 데이터을 제거하거나 변조하면 다시 재 학습을 수행해야 하므로 수행 시간이 오래 걸림**
- 일반적으로 테스트 데이터(검증 데이터)에 특정 feature 들을 반복적으로 변조한 뒤 해당 feature의 중요도를 평균적으로 산정

feature importance는 트리를 나누는 기준이지 Model 에서의 성능을 판단하는 절대적인 기준이 되지 않는다. (물론 많은 연관관계가 있는 것은 맞다) -> 그래서 Permutation importance를 이용하는 것.



- Feature importance는 최적 tree 구조를 만들기 위한 피처들의 impurity가 중요 기준이다. 결정값과 관련이 없어서 feature importance가 높아질 수 있다.
- Feature importance는 학습 데이터를 기반으로 생성됨, Test data에서는 다른 중요도를 나타낼 수 있다.
- Feature importance는 number 형의 높은 cardinality feature에 biased 되어 있다.

출처 및 참고: 머신러닝 완벽 가이드

(https://www.inflearn.com/course/%ED%8C%8C%EC%9D%B4%EC%8D%AC-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-%EC%99%84%EB%B2%BD%EA%B0%80%EC%9D%B4%EB%93%9C#)