



REDWINE


박소현, 김민석, 유연종, 이윤성, 오지승

QUALITY

데이터 출처

kaggle

Red Wine Quality

 Dataset



^

1522

Red Wine Quality

Simple and clean practice dataset for regression or classification modelling




UCI Machine Learning • updated 4 years ago (Version 2)

[Data](#) [Tasks \(4\)](#) [Code \(773\)](#) [Discussion \(13\)](#) [Activity](#) [Metadata](#)

[Download \(99 KB\)](#)

[New Notebook](#)



 Usability 8.8



License Database: Open Database, Contents: Database Contents



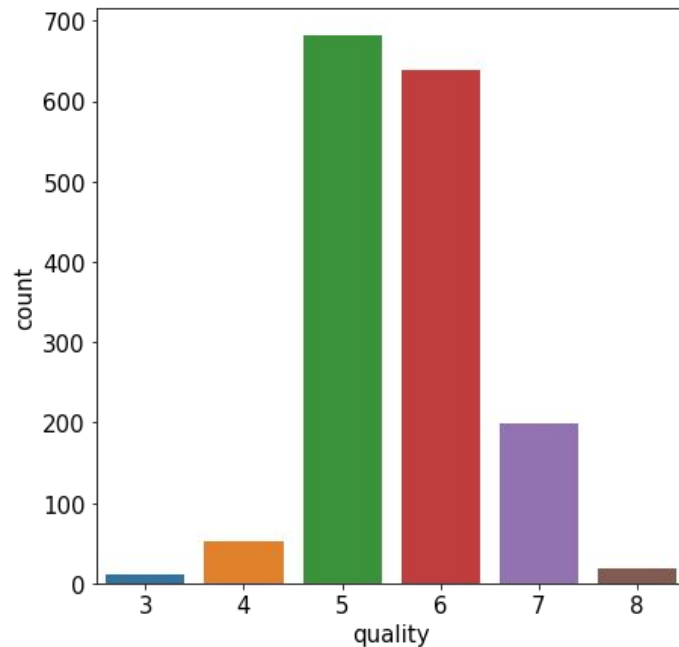
Tags earth and nature, education, beginner, alcohol

데이터 - Red Wine Quality

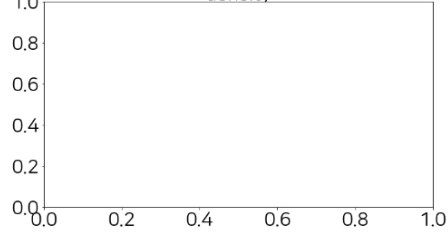
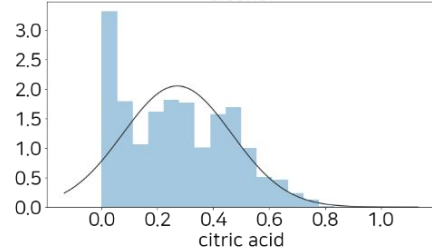
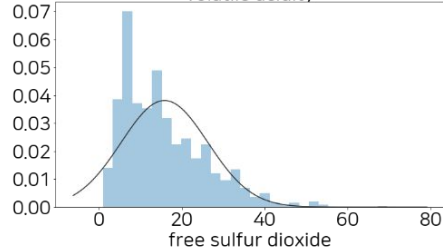
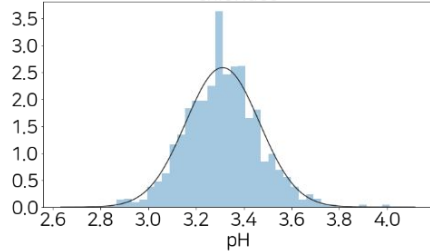
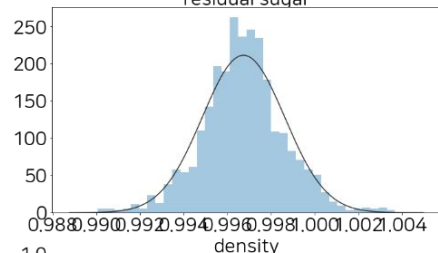
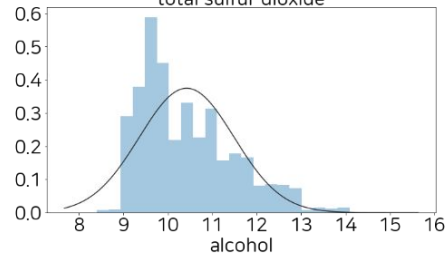
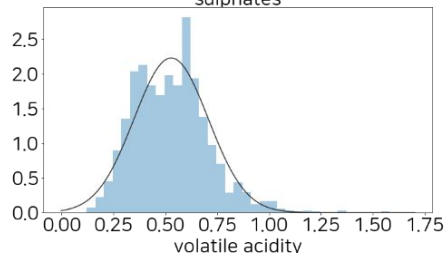
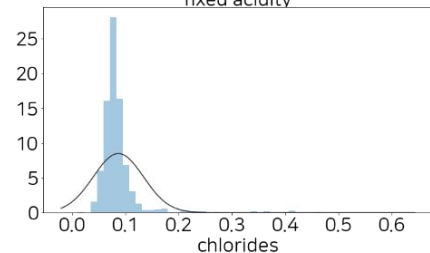
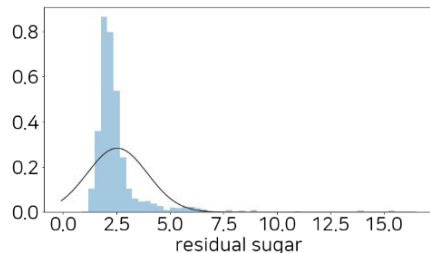
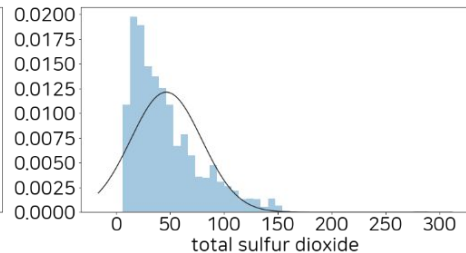
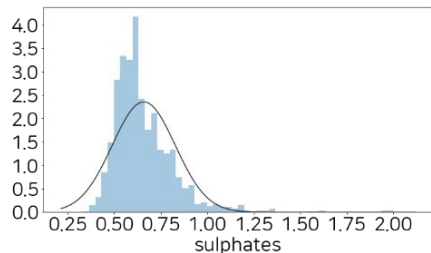
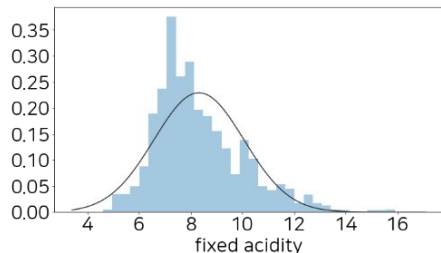
RangeIndex: 1599 entries, 0 to 1598

Data columns (total 12 columns):

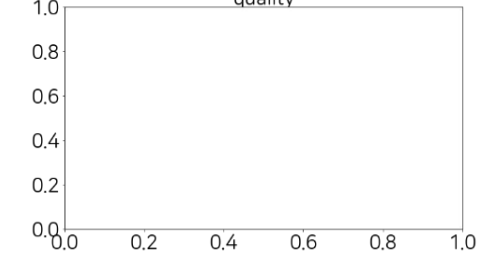
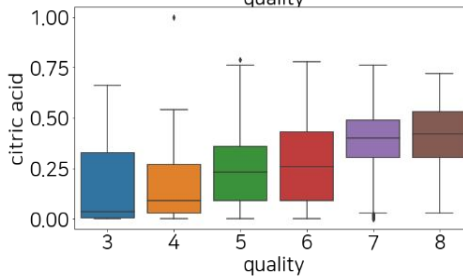
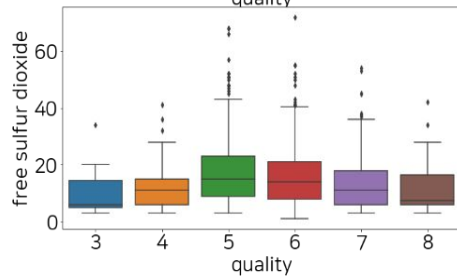
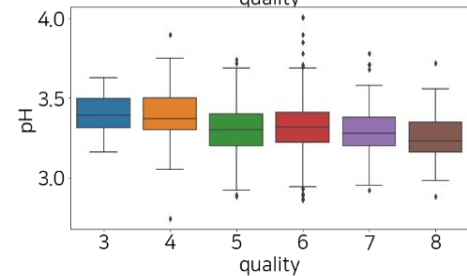
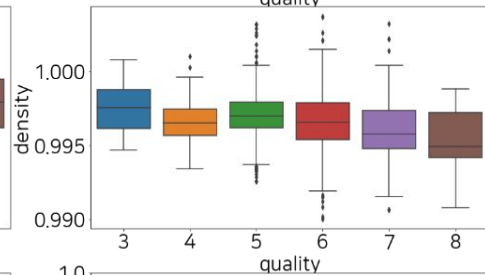
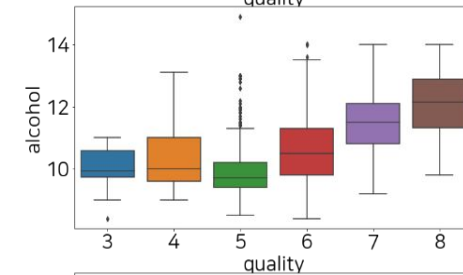
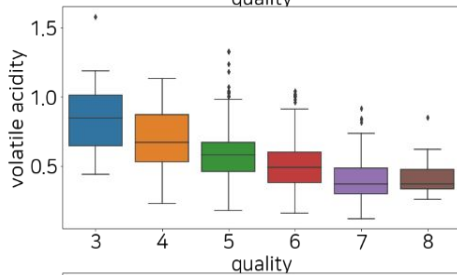
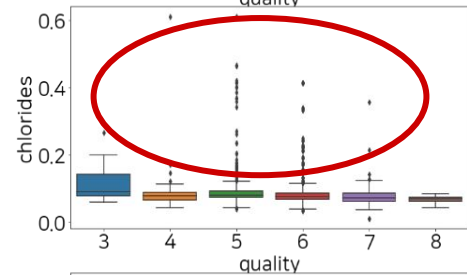
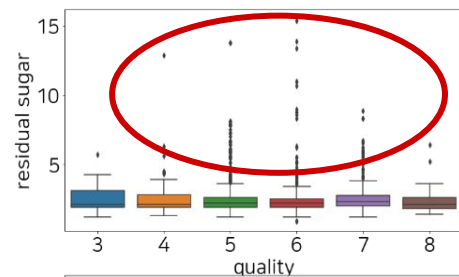
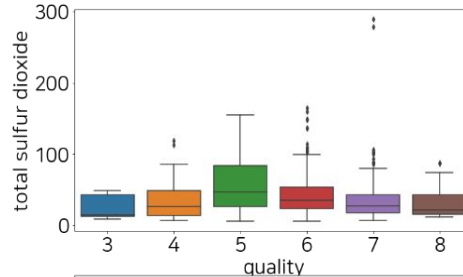
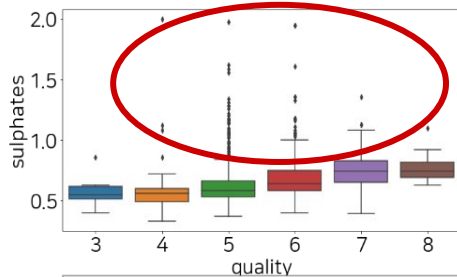
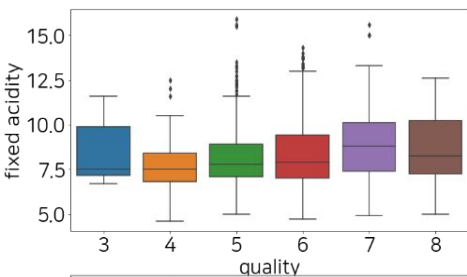
fixed acidity	1599 non-null float64
volatile acidity	1599 non-null float64
citric acid	1599 non-null float64
residual sugar	1599 non-null float64
chlorides	1599 non-null float64
free sulfur dioxide	1599 non-null float64
total sulfur dioxide	1599 non-null float64
density	1599 non-null float64
pH	1599 non-null float64
sulphates	1599 non-null float64
alcohol	1599 non-null float64
quality	1599 non-null int64



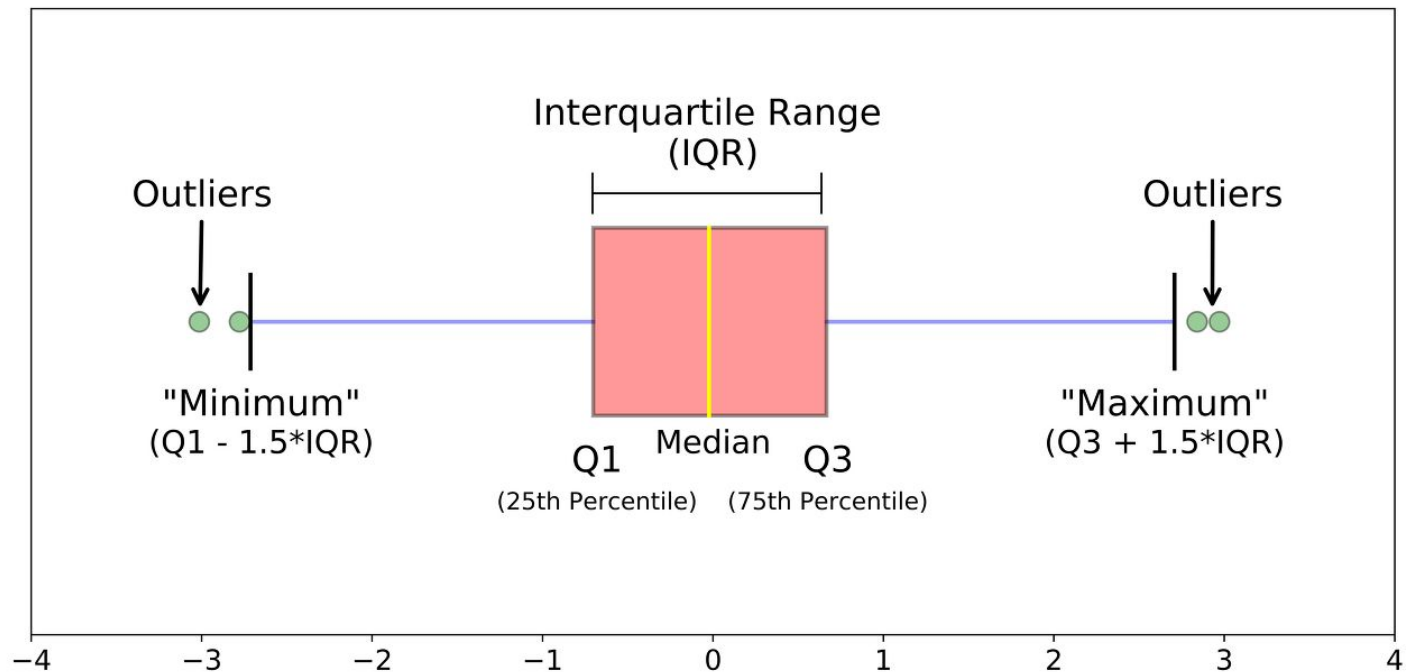
히스토그램



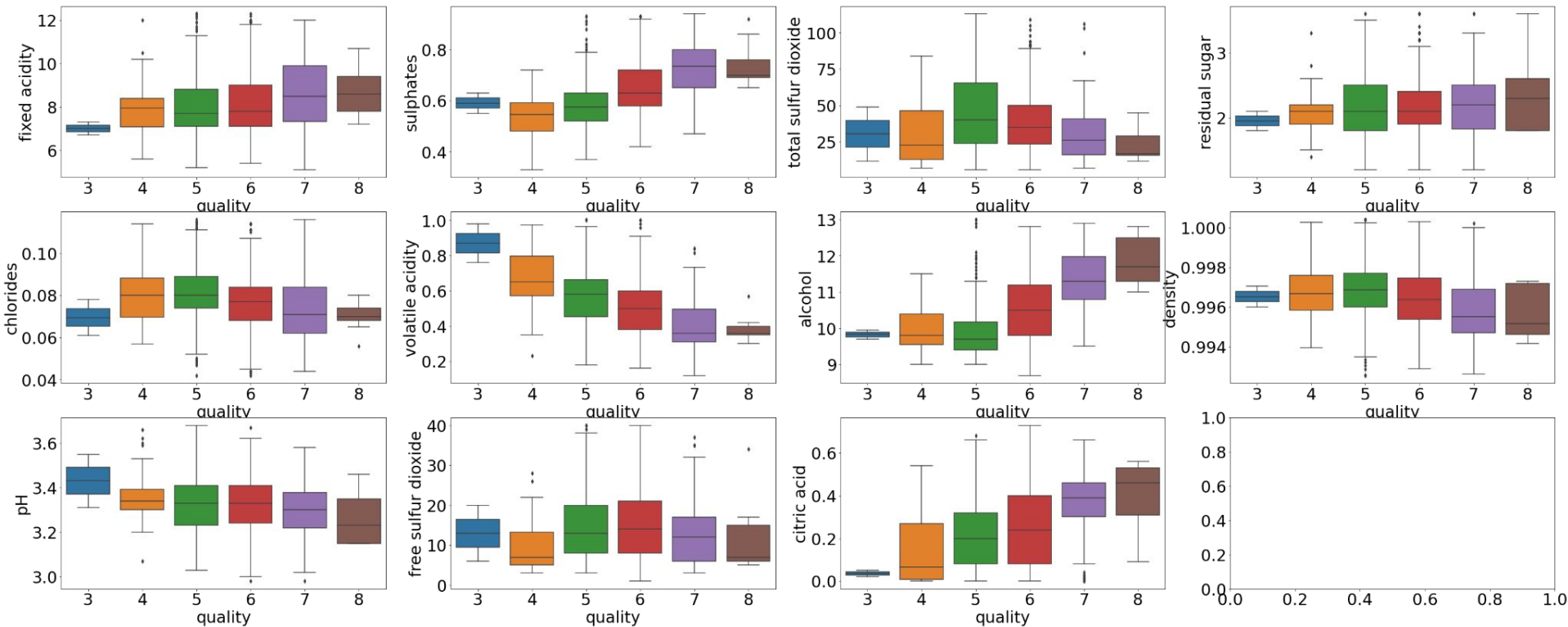
이상치 발견



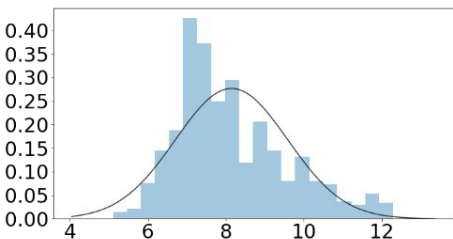
이상치 제거(IQR)



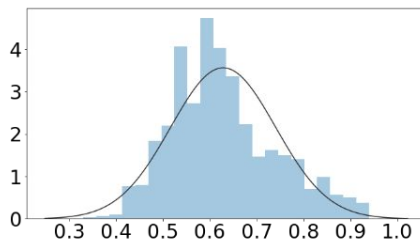
이상치 제거 결과



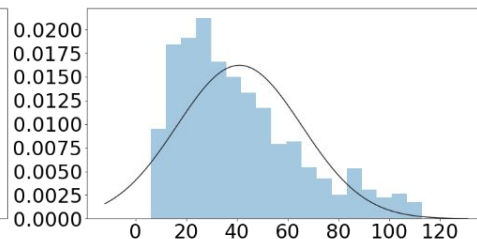
히스토그램



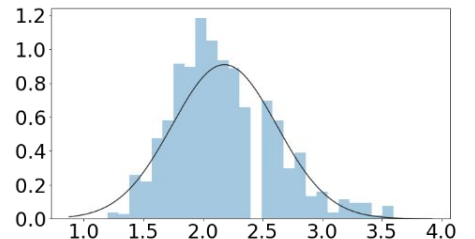
fixed acidity



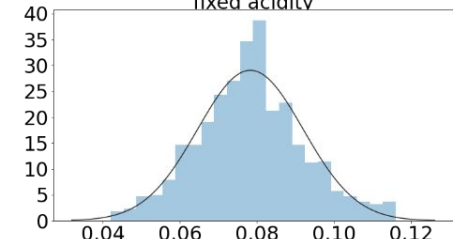
sulphates



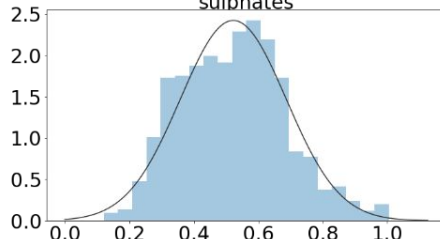
total sulfur dioxide



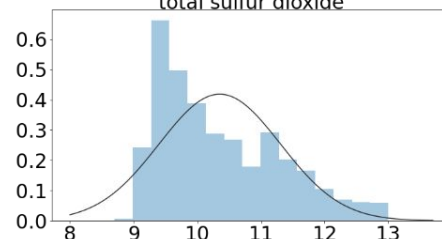
residual sugar



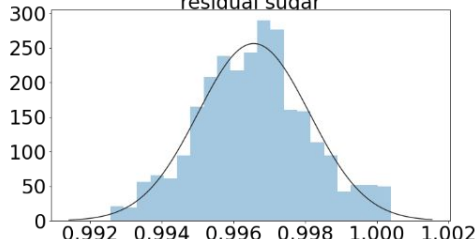
chlorides



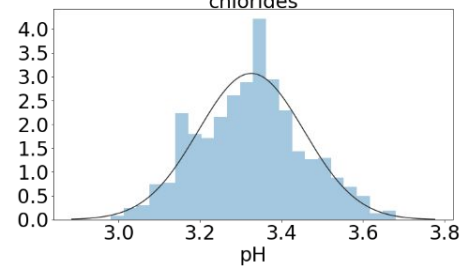
volatile acidity



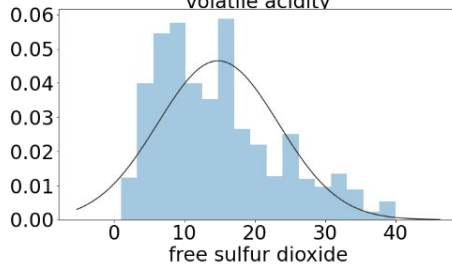
alcohol



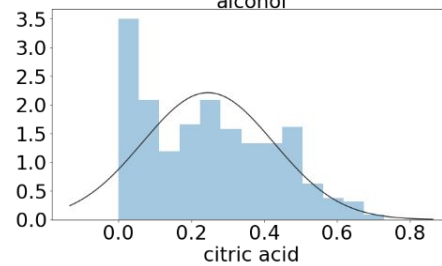
density



pH



free sulfur dioxide

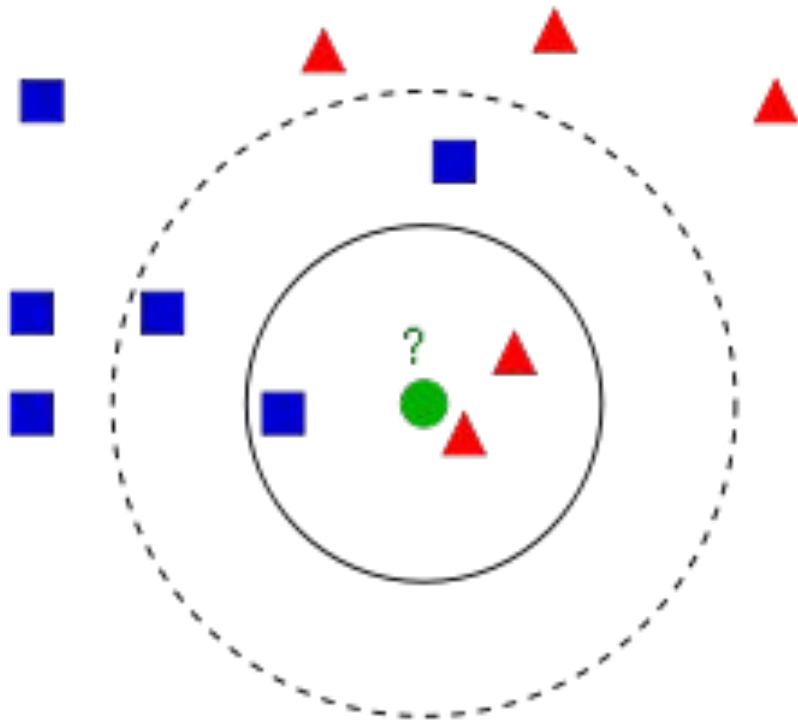


citric acid

VIF 계수 확인

feature	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
VIF Factor	6.8	2.1	3.1	1.7	1.3	1.9	2.1	6.4	3.1	1.3	3.6

K-Nearest Neighbors



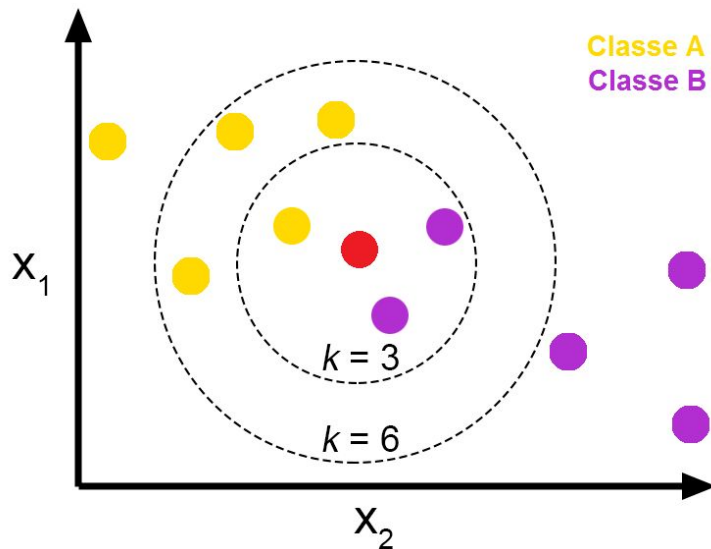
가장 가까운 데이터를 찾아 분류 및 회귀

계으른 학습

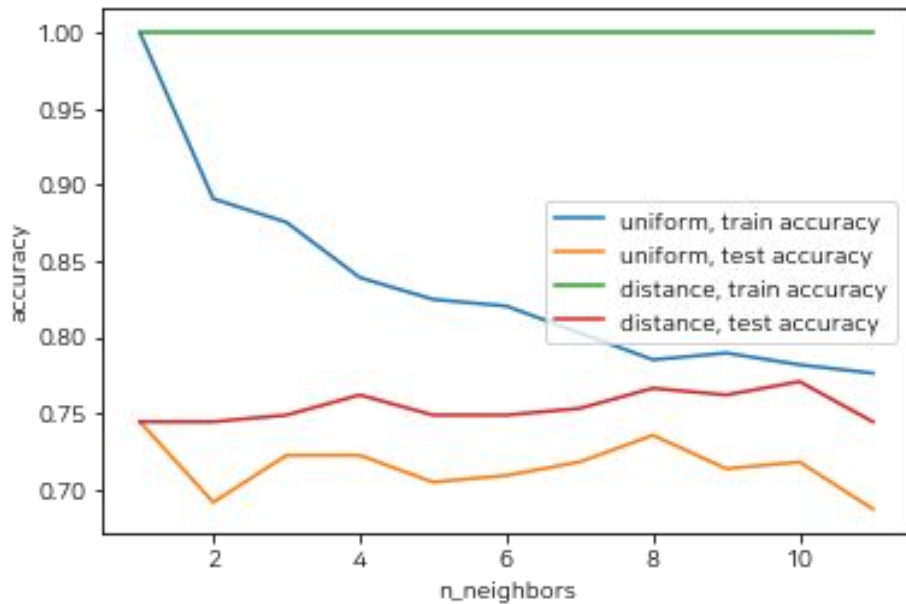
- 데이터 입력 마다 처리

K-Nearest Neighbors

1. 데이터 로드
2. K 값 초기화(`n_neighbors`)
3. 입력된 데이터와 기존 데이터의 거리 계산
4. 가장 짧은 거리순으로 정렬
5. 정렬된 데이터 중 처음부터 순서대로
K개의 기존 데이터를 선정
6. K개의 항목의 레이블 확인
7. 회귀 - 선정된 레이블의 평균 반환
분류 - 선정된 레이블의 모드(0,1)를 반환
- 과반수



n_neighbors, weights



테스트 세트의 정확도 : 0.7048

pre :

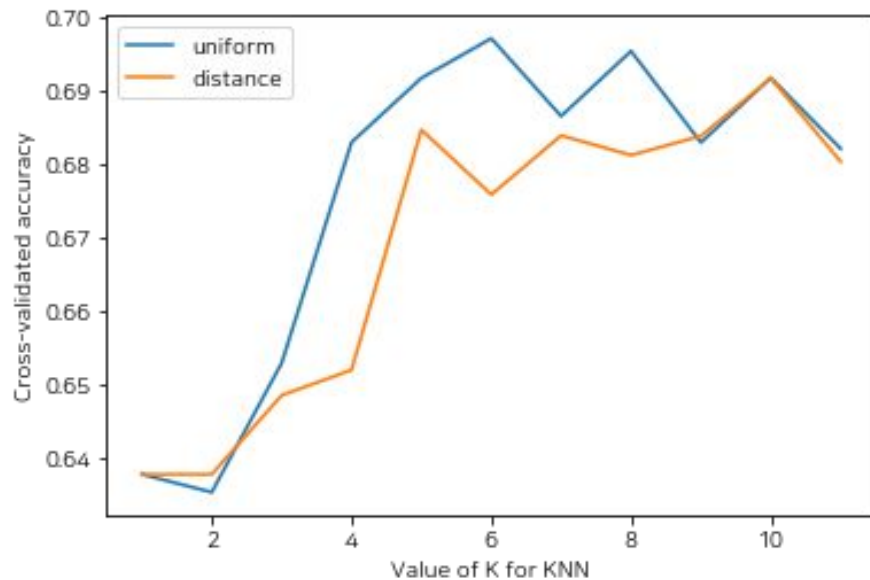
[1 1 0 1 1 1 1 0 0 1]

y :

[1 1 0 0 1 1 1 0 0 1]

n_neighbors = 10

weights = 'distance'



테스트 세트의 정확도 : 0.7709

pre :

[1 1 0 0 1 1 1 0 0 1]

y :

[1 1 0 0 1 1 1 0 0 1]

로지스틱 Logistic

StandardScaler(X) - 분류에 용이, 데이터를 정규 분포로 만듦

RobustScaler(X) - 이상치에 영향을 받지 않음

MinMaxScaler(a,b) - 회귀에 유용

로지스틱 Logistic

LogisticRegression(C=n, random_state=42)

원본 데이터	c =1	c = 0.1	c = 0.01	c =0.001
StandardScaler	0.74	0.74	0.73	0.75
RobustScaler	0.74	0.74	0.73	0.73
MinMaxScaler	0.74	0.74	0.61	0.53

0 . 로지스틱 Logistic

LogisticRegression(C=n, random_state=42)

전처리된 데이터	c = 1	c = 0.1	c = 0.01	c = 0.001
StandardScaler	0.74	0.74	0.74	0.72
RobustScaler	0.74	0.74	0.74	0.70
MinMaxScaler	0.75	0.73	0.68	0.53

C = 1
이상치 제거 했을 때
MinMaxScaler 가장 좋음



MinMaxScaler 선택

로지스틱 Logistic

- 검증세트 `cross_val_score` 사용

train_data_pred	test_data_pred	교차 검증 평균 점수
0.74	0.75	0.73

예측률은 낮지만 균일한 모습

로지스틱 Logistic

데이터 특성을
제거하는
경우



오히려 예측률 감소

0 . 로지스틱 Logistic

GridSearchCV 과정

C
solver



C =1
saga

0.74

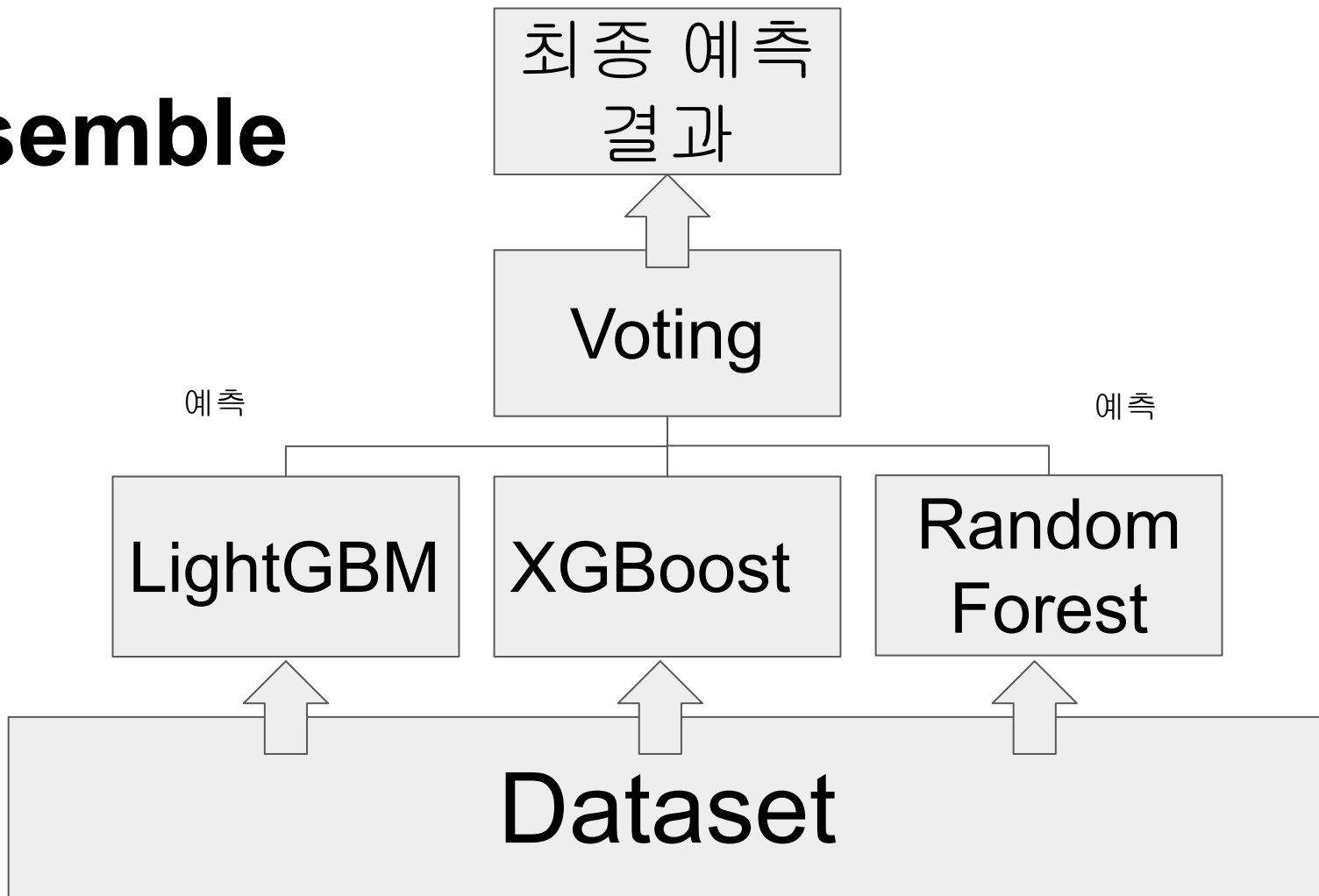
로지스틱 Logistic

`LogisticRegression(random_state=42)` 일 때

데이터 특성에 변화를 주지 않을 때

`test_data` 의 예측률 0.75로 가장 높은 값

Ensemble



앙상블 학습 유형

bagging

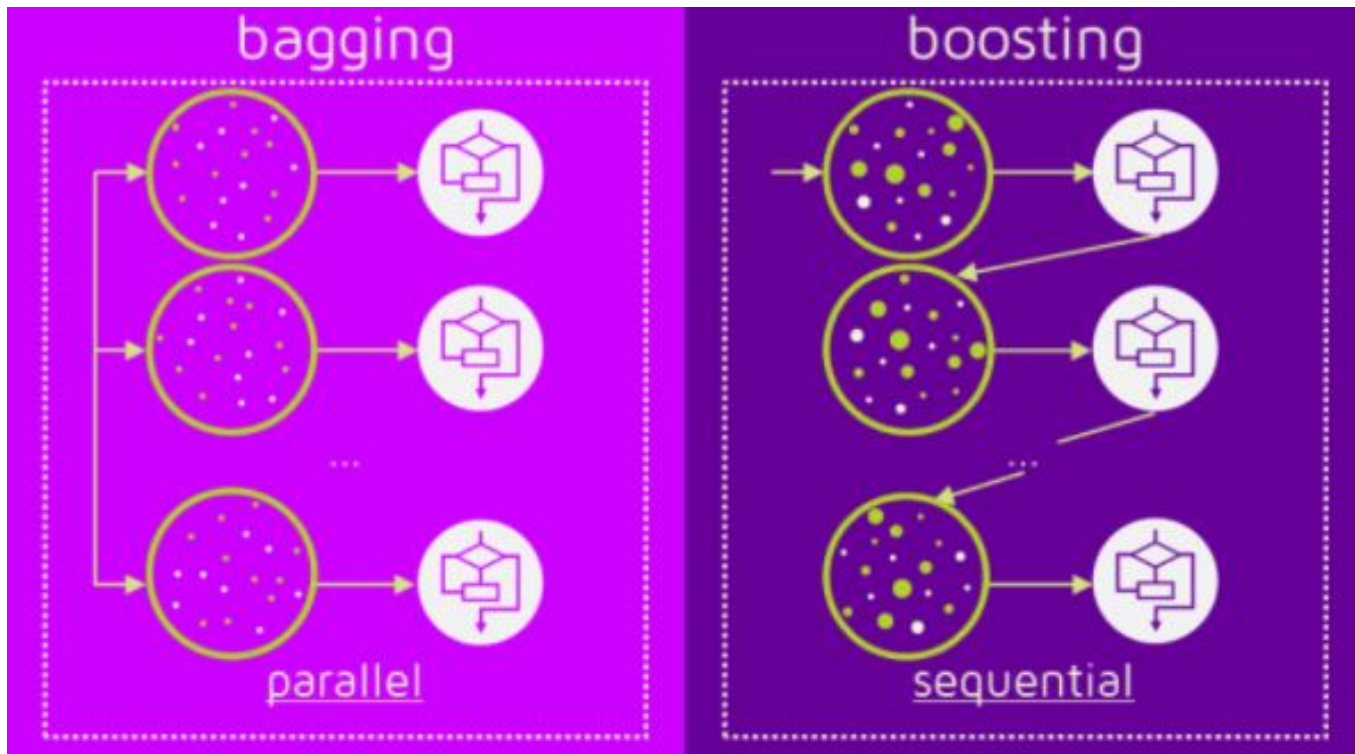
데이터를 분류한 후
각각 데이터를
학습시키고 결과를 집계

Random Forest

boosting

분류한 후 올바르게
예측할 수 있게 다음
분류기에 가중치 부여

Xgboost, LightGBM



파라미터 최적값 탐색

GridSearchCV()

```
params_ = { 'n_estimators' : range(1, 1001,100),  
            'max_depth' : [10, 50, 100, 200, 300],  
            'eta' : [0.05 , 0.07, 0.09 ,1],  
            'booster' : ['rb','gbdt','dart','goss']  
            }  
  
params = { 'n_estimators' : range(1, 1001,100),  
            'max_depth' : [10, 50, 100, 200, 300],  
            'min_samples_leaf' : [4,8, 12, 16, 20],  
            'min_samples_split' : [4, 8,12, 16, 20]  
            }  
  
rf_clf = XGBClassifier(random_state = 42, n_jobs = -1)  
grid_cv = GridSearchCV(rf_clf, param_grid = params, cv = 3, n_jobs = -1)  
grid_cv.fit(X_train, y_train)
```

스케일링 종류별 예측도 변화

Standard

0.7918

MinMax

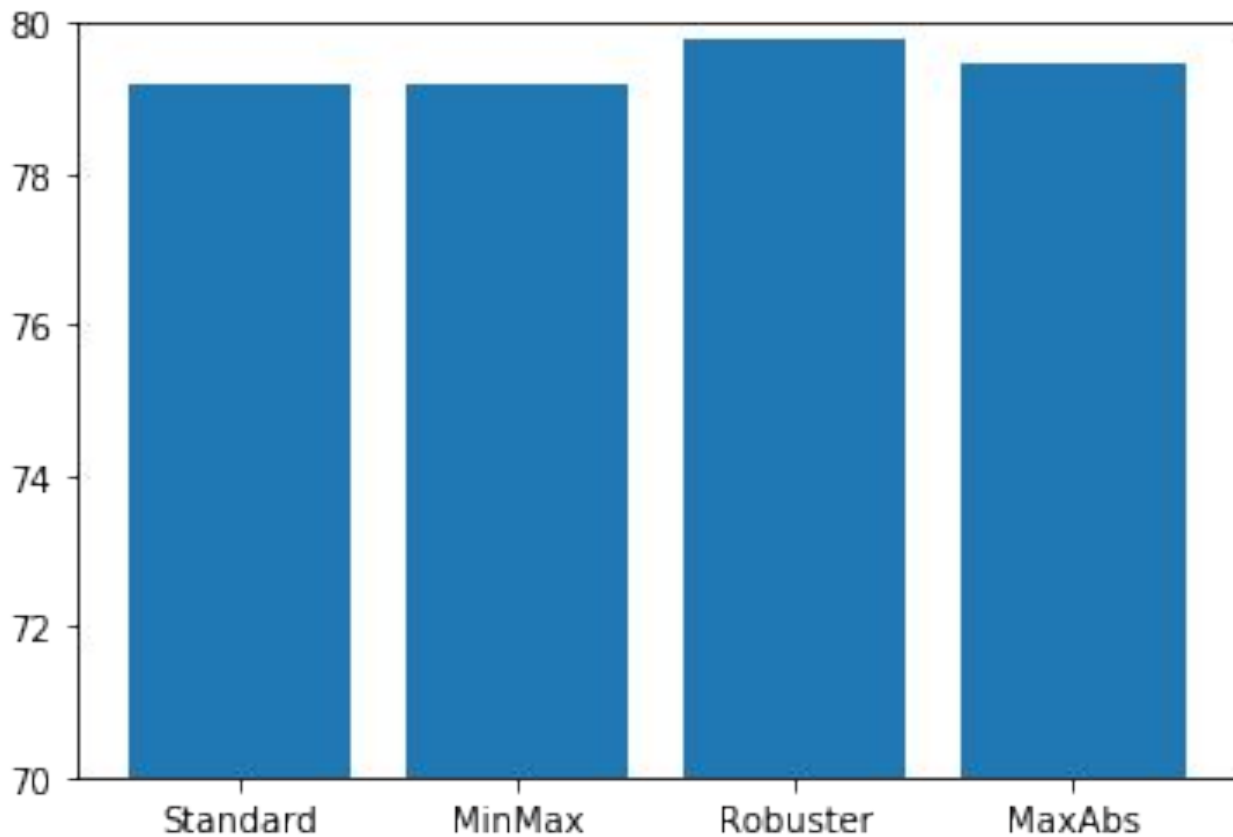
0.7918

Robuster

0.7977

MaxAbs

0.7947

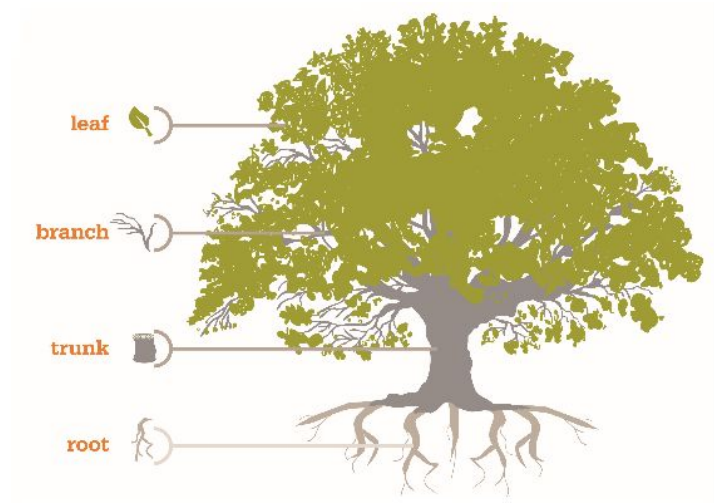


Decision Tree (의사결정나무)

1. 의사결정 나무란 ?

주어진 입력값에 대하여 출력값을 예측하는 모형

- 나무형태의 그래프로 표현
- 예측력은 다른 지도학습 기법들에 비해 대체로 떨어지나 해석력이 좋음

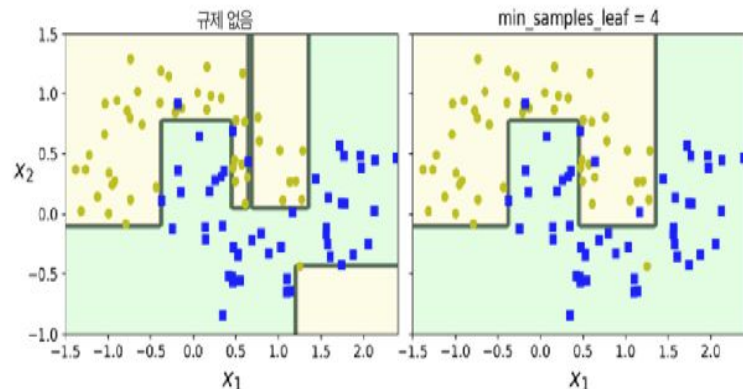


2. 선택

1. 불순도(Impurity) criterion 매개변수

지니 계수(Gini Index)

엔트로피(Entropy)



- **max_depth** : 결정 트리의 최대 깊이

- **min_samples_split** : 분할되기 위해 노드가 가져야 하는 최소 샘플 개수

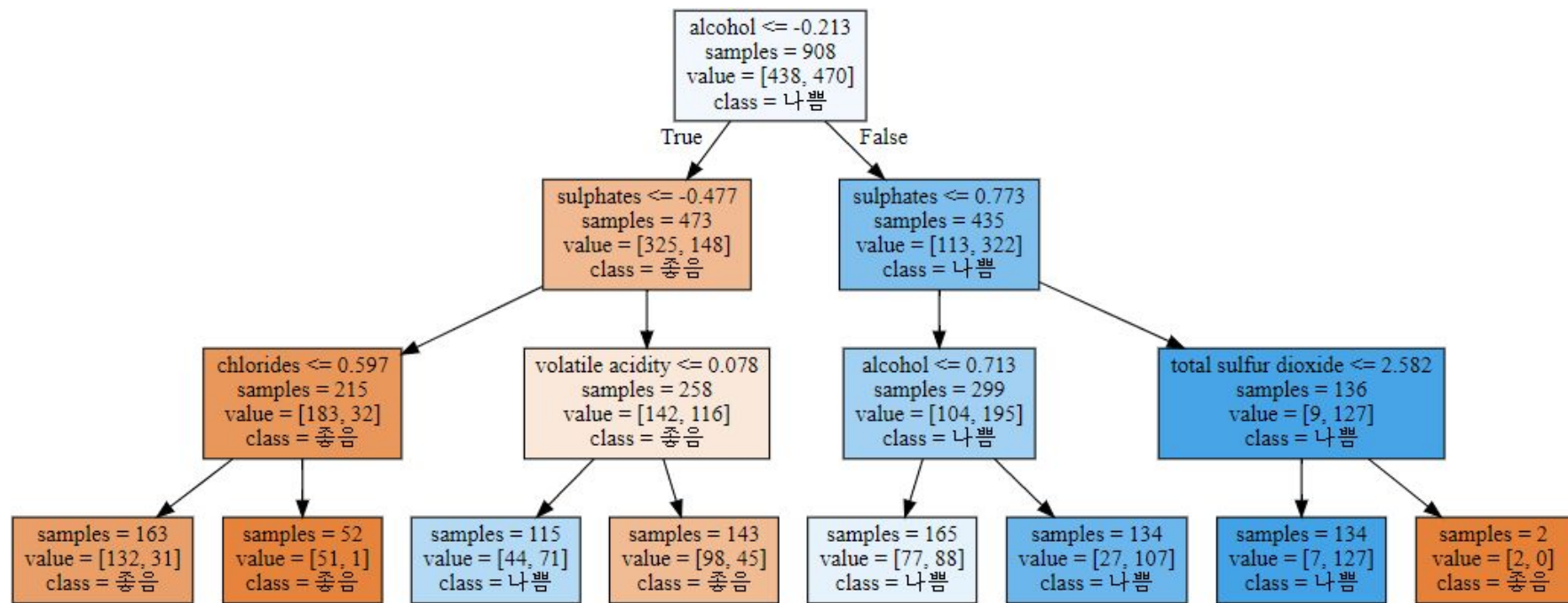
- **min_samples_leaf** : 리프 노드가 가지고 있어야 할 최소 샘플 개수

- **min_weight_fraction_leaf** : min_samples_leaf와 같지만, 가중치가

부여된 전체 샘플 수에서의 비율

- **max_leaf_nodes** : 리프 노드의 최대 개수

-- **max_features** : 각 노드에서 분할에 사용할 특성의 최대 개수



3. 결과

entropy	0.73
max_depth =4	0.72

gini	0.70
max_depth =4	0.71

결론

과대적합이 쉽다

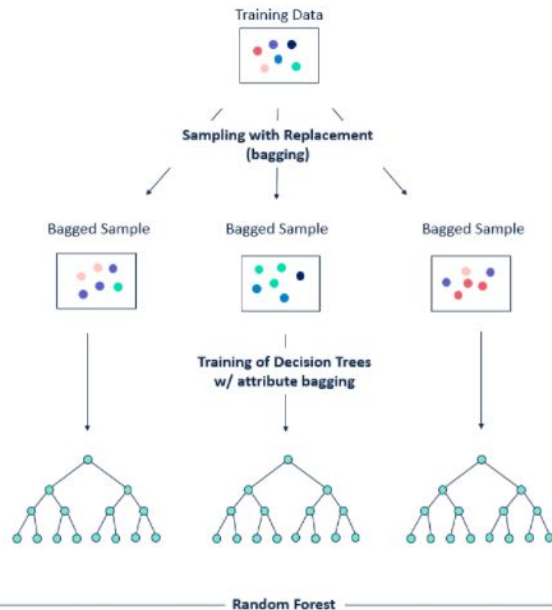
Random Forest(랜덤 포레스트)

장점

- 결정 트리의 쉽고 직관적인 장점을 그대로 가지고 있음
- 앙상블 알고리즘 중 비교적 빠른 수행 속도를 가지고 있음
- 다양한 분야에서 좋은 성능을 나타냄

단점

- 하이퍼 파라미터가 많아 튜닝을 위한 시간이 많이 소요됨



2. 선택

- **n_estimators** : 결정트리의 갯수를 지정
- **min_samples_split** : 분할되기 위해 노드가 가져야 하는 최소 샘플 개수
- **min_samples_leaf** : 리프 노드가 가지고 있어야 할 최소 샘플 개수
- **max_feature** : 최적의 분할을 위해 고려할 최대 feature 개수
- **max_depth** : 트리의 최대깊이
- **max_leaf_nodes** : 리프 노드의 최대 개수
- **max_features** : 각 노드에서 분할에 사용할 특성의 최대 개수

3. 결과

n_estimators = 450	0.76
max_depth =4	0.76

결론

3. 결과 -1(GridSearchCV)

```
1 from sklearn.model_selection import GridSearchCV
2
3 params = { 'n_estimators' : [10, 100],
4           'max_depth' : [6, 8, 10, 12],
5           'min_samples_leaf' : [8, 12, 18],
6           'min_samples_split' : [8, 16, 20]
7         }
8
9 # RandomForestClassifier 객체 생성 후 GridSearchCV 수행
10 rf_clf = RandomForestClassifier(random_state = 0, n_jobs = -1)
11 grid_cv = GridSearchCV(rf_clf, param_grid = params, cv = 3, n_jobs = -1)
12 grid_cv.fit(X_train, y_train)
13
14 print('최적 하이퍼 파라미터: ', grid_cv.best_params_)
15 print('최고 예측 정확도: {:.4f}'.format(grid_cv.best_score_))
```

최적 하이퍼 파라미터: {'max_depth': 10, 'min_samples_leaf': 8, 'min_samples_split': 8, 'n_estimators': 10}
최고 예측 정확도: 0.7698

svm 이란?

1. svm이란 1970년 초반 러시아의 과학자 블라디미르가 제안하였지만 초반에는 인기를 얻지 못하였다.
2. 1990년대 들어 분류문제에서 svm의 우수성이 입증되었고 특히 머신러닝 알고리즘에서 인기있는 모델
3. SVM은 선형 또는 비선형 분류 뿐만아니라 회귀, 이상치 탐색에도 사용할 수 있는 모델이며, 특히 복잡한 분류 문제에 잘 맞으며, 중간 크기의 데이터셋에 적합하다.
4. 웬만한 상황에서 딥 러닝 못지 않은 성능을 내고, 무엇보다도 가볍기 때문이다.

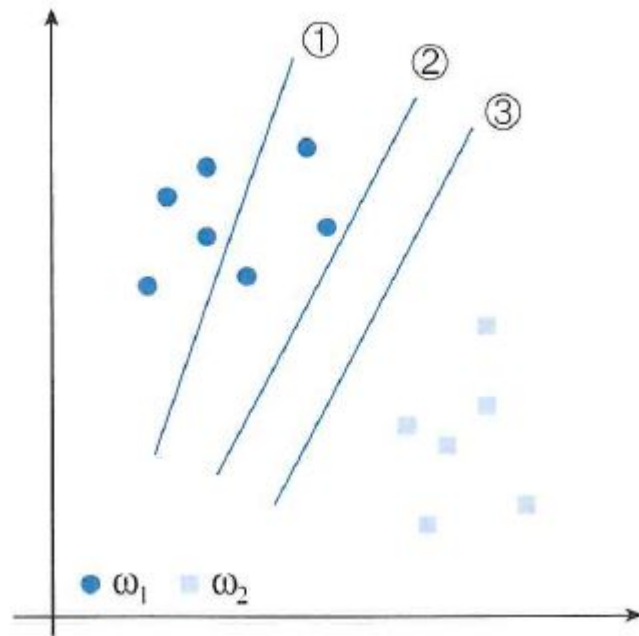
svm 이란?

① 분류기는 **Train set**을 틀리게 분류한다.

이를 여러번 학습시켜 모델링하면 ②와 ③ 분류기와 같이 될것이다.

Train set 측면에서 보면 ②와 ③ 분류기는 오류가 0이므로 같은 성능을 가진 분류기로 볼 수 있다. 하지만, 일반화(**generalization**) 측면에서 보면 ② 보다 ③이 더 낫다고 할 수 있다. 그 이유는 ③ 분류기가 두 개의 클래스에 대해 여백(**margin**) 크기 때문이다.

바로 여기서 이러한 여백, 즉 마진을 어떻게 공식화하고 이 마진을 최대화하는 결정 초평면(**decision hyperplane**)을 찾는 것이 바로 **SVM**의 발상이라 할 수 있다



하이퍼 파라미터 튜닝결과 시각화

kernel

Linear

Accuracy: 0.7268722466960352

poly

Accuracy: 0.7533039647577092

rbf

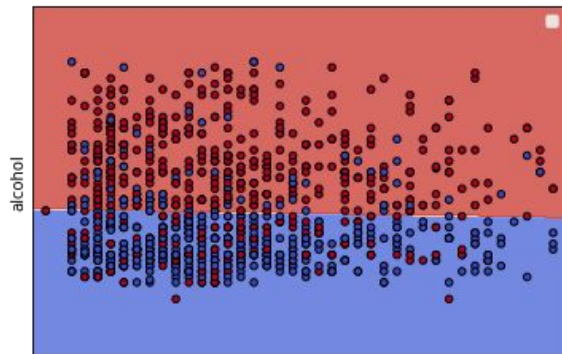
Accuracy: 0.8105726872246696

sigmoid

Accuracy: 0.5374449339207048

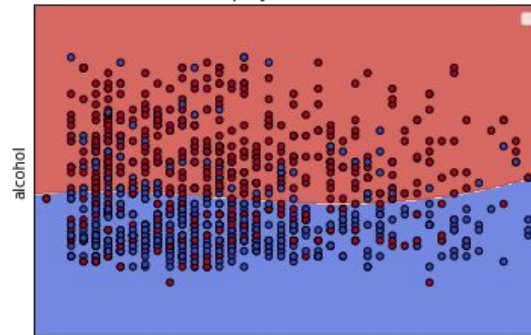
Linear

linear of SVC



poly

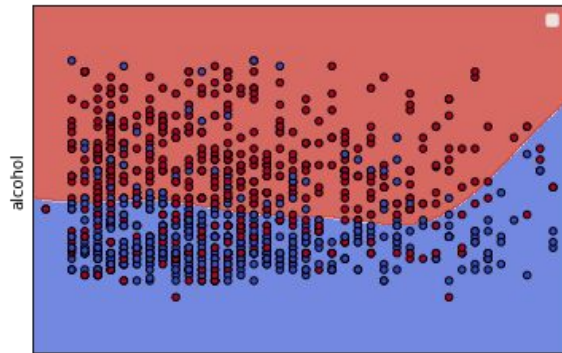
poly of SVC



rbf

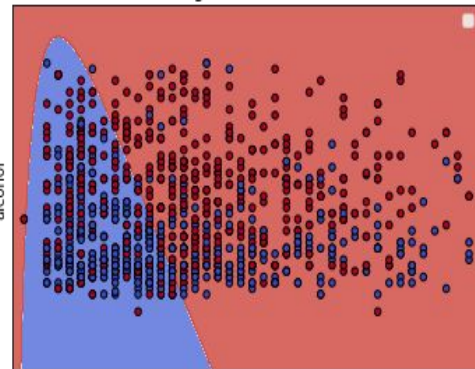
free sulfur dioxide

rbf of SVC



sigmoid

sigmoid of SVC



svm의 튜닝값

```
clf = svm.SVC(C = 10, kernel='rbf', gamma = 13)
```

Accuracy: 0.8105726872246696

Result

SVM	81
Ensemble	79
KNN	77.09
Random Forest	76.9
Logistic	75

Result

아쉬운 점
과 적 합
인공 신경망

참고

Towards Data Science - [Understanding Boxplots](#), [K-Nearest Neighbors](#)

Medium - [Distance Metrics](#)

wikipedia - [K-최근접 이웃 알고리즘](#)

SVM - [서포트벡터머신\(SVM\)](#)