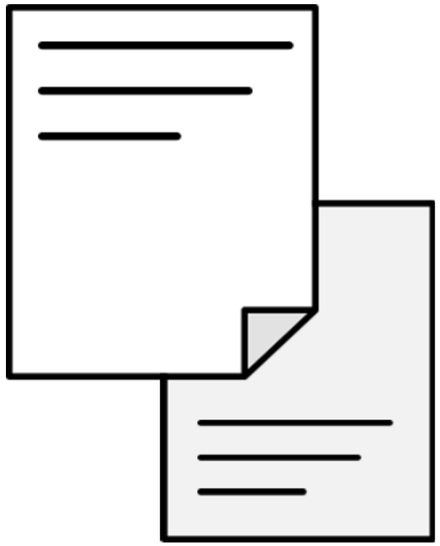
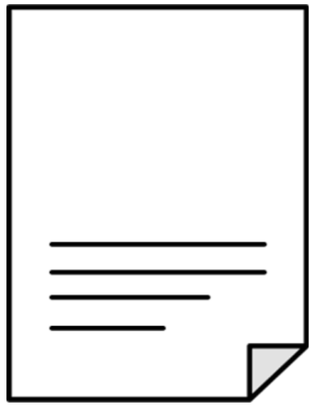


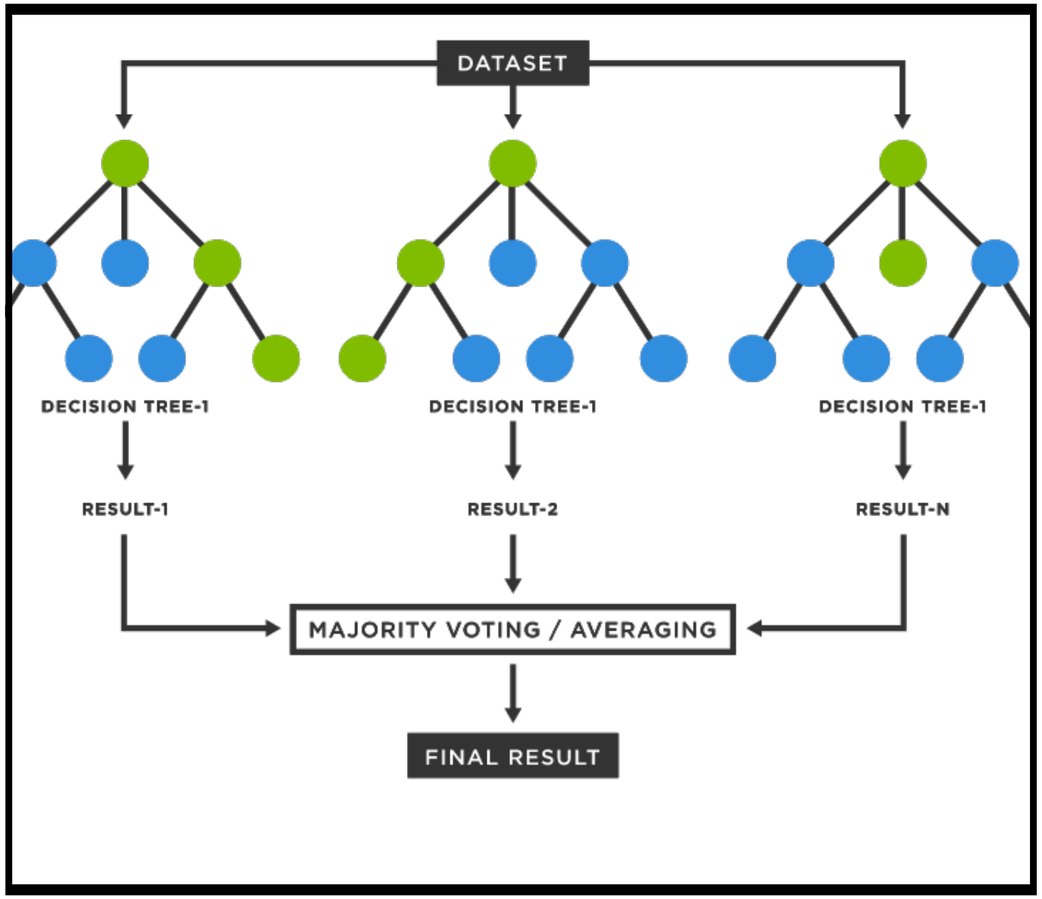
Random Forest와 Gradient Boosting



발표자 : 최강규

CONTENTS

SECURIOUS



01

RANDOM FOREST

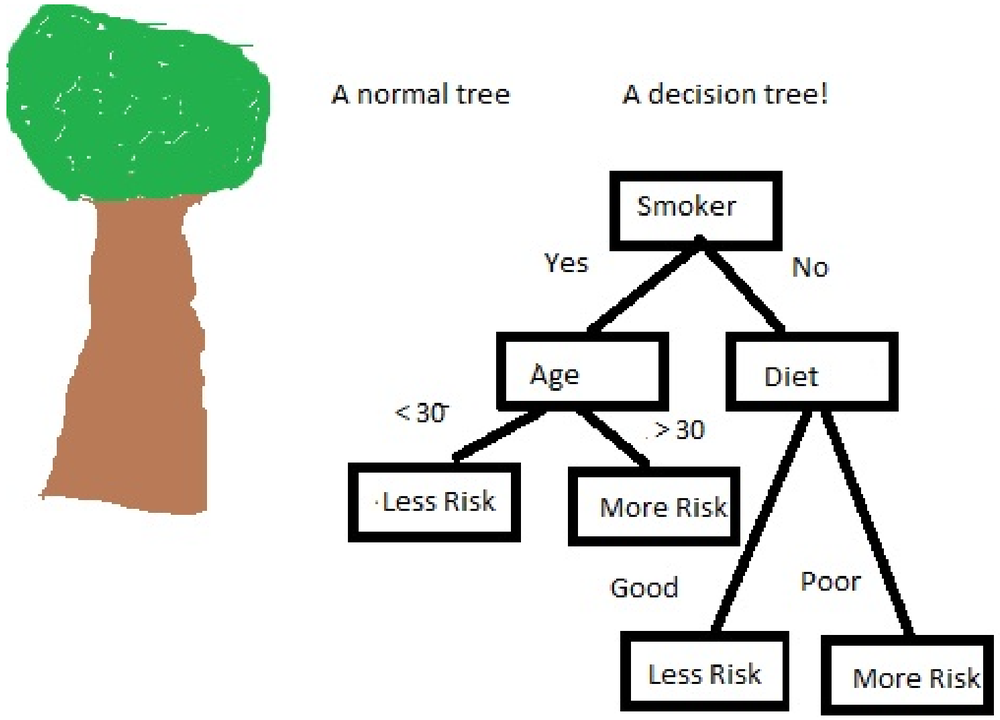
분류, 회귀 분석 등에 사용되는 앙상블 학습 방법의 일종으로, 훈련 과정에서 구성한 다수의 결정 트리로부터 부류(분류) 또는 평균 예측치(회귀 분석)를 출력함으로써 동작한다.

SECURIOUS

01 RANDOM FOREST

DECISION TREE 복습

의사 결정 트리는, 특정 Feature 에 대한 질문을 기반으로 데이터를 분리하는 방법



Ex) 건강위험도를 결정하는 의사결정 트리

어떠한 사람에 대한 정보(feature)가 주어졌을 때, “흡연자인지, 몇살인지, 다이어트는 하고 있는지” 등의 질문을 통해, 해당 사람을 논리적으로 분리

RANDOM FOREST

Random Forest와 Decision Tree의 차이점

Random Forest: 수많은 의사 결정 트리(Decision Tree)가 모여서 생성

Decision Tree: 건강 위험도를 세 가지 요소와 한가지 의사 결정 트리로 인해서 결정

하지만, 건강 위험도를 예측하려면 세 가지 요소보다 더 많은 요소를 고려하는 것이 바람직할 것입니다.

ex) 성별, 키, 몸무게, 거주지역, 운동량, 기초 대사량, 근육량 등 수많은 요소도 건강에 큰 영향을 미칩니다.

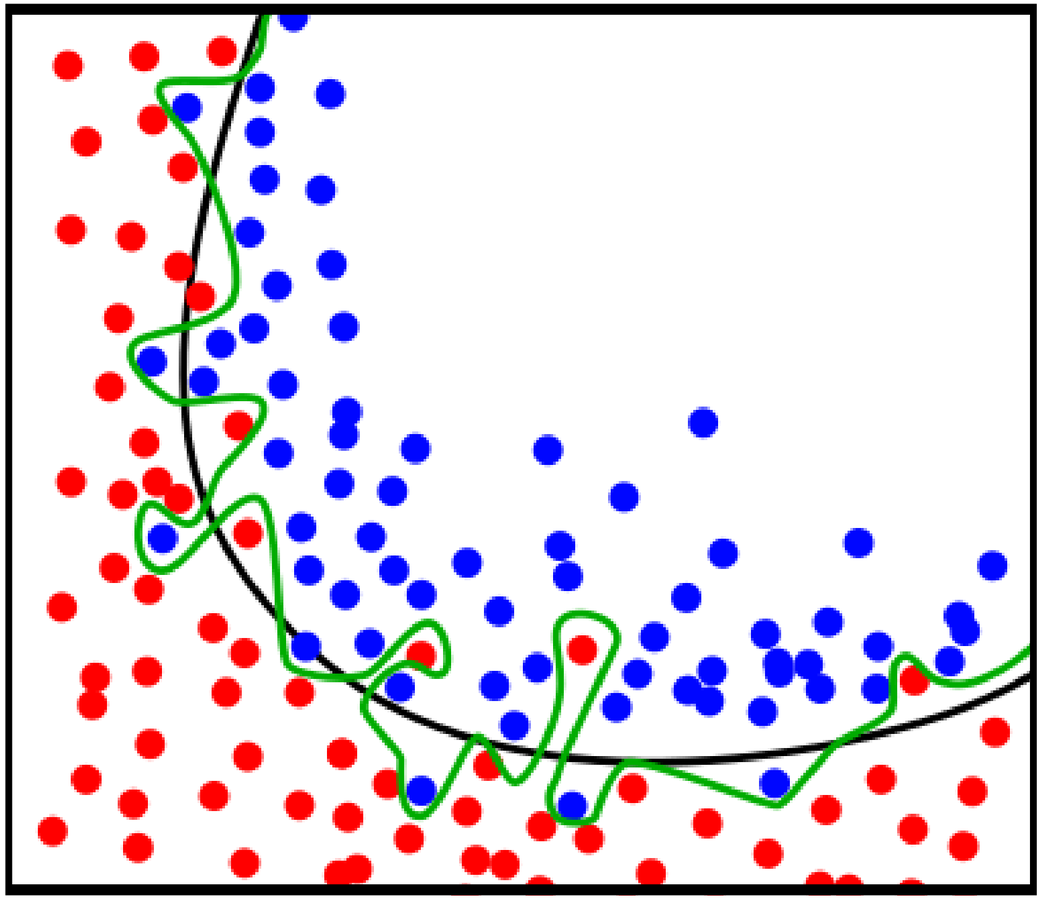
RANDOM FOREST

Random Forest 오버피팅

수많은 요소(Feature)들을 기반으로 건강 위험도(Target)를 예측한다면?

-> 오버피팅 발생

Ex) Feature가 30개라고 가정
30개의 Feature를 기반으로 하나의 결정 트리를 만든다면 트리의 가지가 많아질 것이고, 이는 오버피팅이 될 것입니다. 따라서, Random Forest 는 전체 Feature 중 랜덤으로 일부 Feature만 선택해 하나의 결정 트리를 만들고, 또 전체 Feature 중 랜덤으로 일부 Feature를 선택해 또 다른 결정 트리를 만들며, 여러 개의 의사 결정 트리를 만드는 방식으로 구성됩니다. (의사 결정 트리마다 하나의 예측 값을 내놓습니다.)



01

OVERFITTING 과적합

학습 데이터를 과하게 학습(overfitting)하는 것을 뜻한다. 일반적으로 학습 데이타는 실제 데이터의 부분 집합이므로 학습데이터에 대해서는 오차가 감소하지만 실제 데이타에 대해서는 오차가 증가하게 된다.

SECURIOUS

RANDOM FOREST

Random Forest의 장점

- Classification(분류) 및 Regression(회귀) 문제에 모두 사용 가능

- Missing value(결측치)를 다루기 쉬움

- 대용량 데이터 처리에 효과적

- 모델의 노이즈를 심화시키는 Overfitting(오버피팅) 문제를 회피하여, 모델 정확도를 향상시킴

- Classification 모델에서 상대적으로 중요한 변수를 선정 및 Ranking 가능

02 코드 및 파라미터

n_estimators

랜덤 포레스트 안의
결정 트리 갯수

- n_estimators는 클수록 좋음.
- 메모리와 훈련 시간이 증가
- Default는 10

max_features

무작위로 선택할
Feature의 개수

max_features 값이 클수록 랜덤 포레스트의 트리들이 매우 비슷해짐.
max_features 값이 작을수록 오버피팅이 줄어듬.
max_features는 일반적으로 Defalut 값을 사용

max_depth

트리의 깊이

min_samples_leaf

리프노드가 되기 위한
최소한의 샘플 데이터 수

SECURIOUS

코드 및 파라미터

분류 코드 (RandomForestClassifier)

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

classifier = RandomForestClassifier(n_estimators = 100)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print("정확도 : {}".format(accuracy_score(y_test, y_pred))
```

코드 및 파라미터

회귀 코드 (RandomForestRegressor)

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

regressor = RandomForestRegressor()

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

mse = np.sqrt(mean_squared_error(y_pred, y_test))

print('평균제곱근오차 : ', mse)
```

SECURIOUS

코드 및 파라미터

GridSearchCV를 통한 랜덤포레스트분류의 하이퍼 파라미터 튜닝

```
from sklearn.model_selection import GridSearchCV

grid = {
    'n_estimators' : [100,200],
    'max_depth' : [6,8,10,12],
    'min_samples_leaf' : [3,5,7,10],
    'min_samples_split' : [2,3,5,10]
}

classifier_grid = GridSearchCV(classifier, param_grid = grid, scoring="accuracy", n_jobs=-1, verbose =1)

classifier_grid.fit(X_train, y_train)

print("최고 평균 정확도 : {}".format(classifier_grid.best_score_))
print("최고의 파라미터 :", classifier_grid.best_params_)
```

코드 및 파라미터

파라미터 중요도 시각화

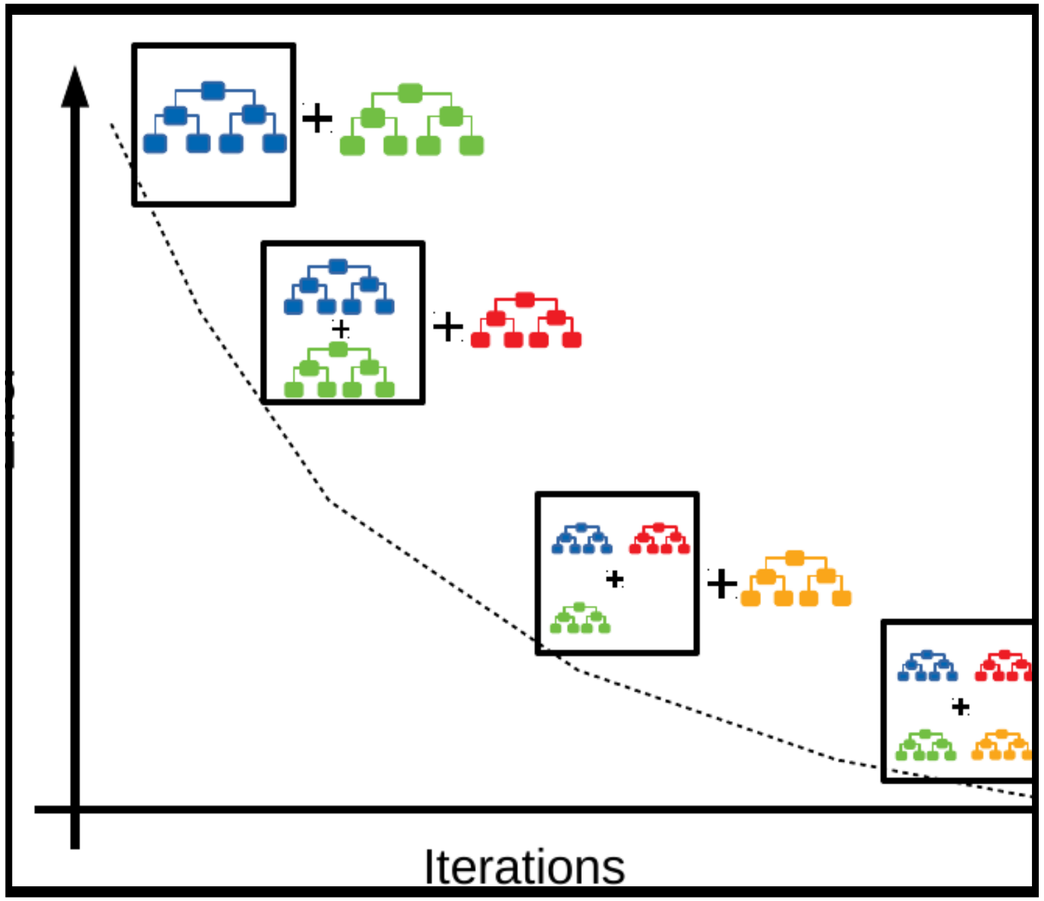
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

feature_importances = model.feature_importances_

ft_importances = pd.Series(feature_importances, index = X_train.columns)
ft_importances = ft_importances.sort_values(ascending=False)

plt.figure(fig.size(12,10))
sns.barplot(x=ft_importances, y= X_train.columns)
plt.show()
```

SECURIOUS



03

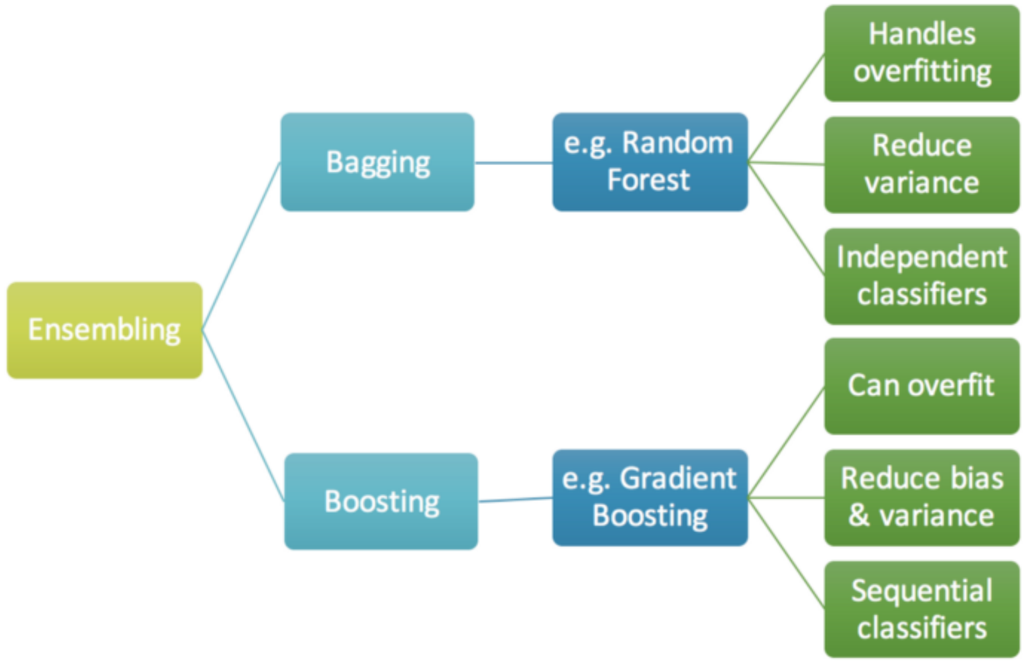
Gradient Boosting

Gradient (잔차)를 이용하여 이전 모형의 약점을 보완하는 새로운 모형을 순차적으로 적합한 뒤 이들을 선형 결합하여 얻어진 모형을 생성하는 지도학습 알고리즘

03 Gradient Boosting

Boosting이란?

Boosting이란 약한 분류기를 결합하여 강한 분류기를 만드는 과정



앙상블 알고리즘의 기본 원리: 각각 0.3 정도의 accuracy를 보여주는 분류기 A, B, C를 결합하여 더 높은 정확도 (0.7 정도의 accuracy)를 얻는 것

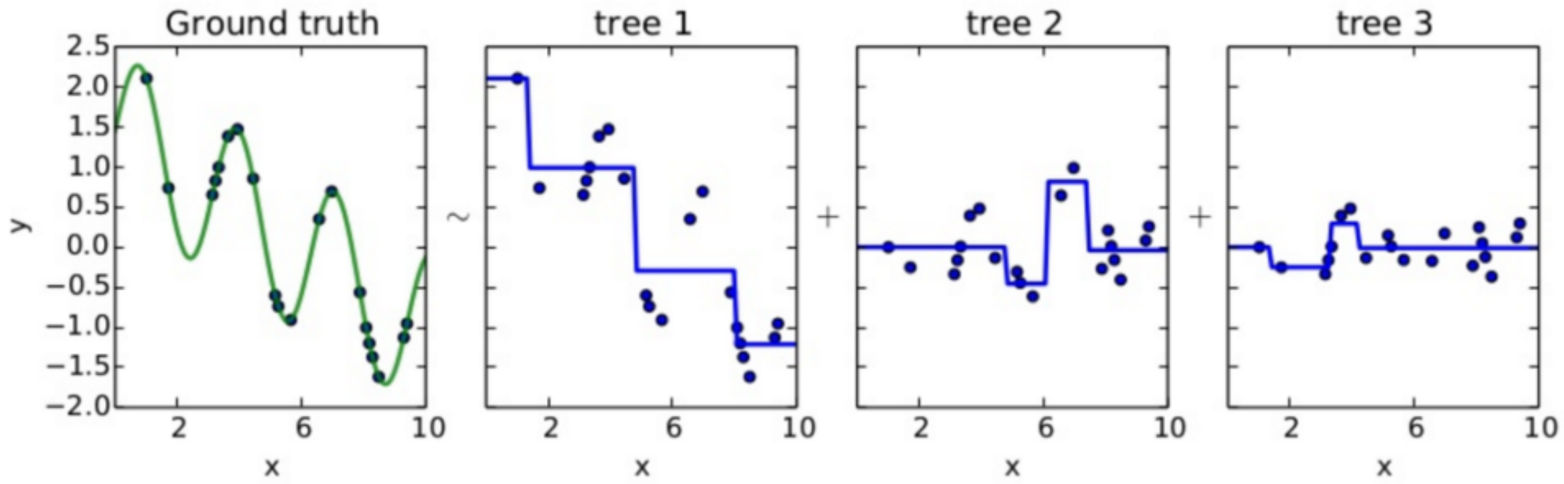
Boosting은 이 과정을 순차적으로 실행합니다. A 분류기를 만든 후, 그 정보를 바탕으로 B 분류기를 만들고, 다시 그 정보를 바탕으로 C 분류기를 만듭니다. 그리고 최종적으로 만들어진 분류기들을 모두 결합하여 최종 모델을 만드는 것이 Boosting의 원리입니다.

SECURIOUS

03 Gradient Boosting

Gradient Boosting 알고리즘

Residual fitting



아주 간단한 모델 A를 통해 y를 예측하고 남은 잔차 (residual)을 다시 B라는 모델을 통해 예측하고 A+B 모델을 통해 y를 예측한다면 A보다 나은 B 모델을 만들 수 있게 되죠. 이러한 방법을 계속하면 잔차는 계속해서 줄어들게되고, training set을 잘 설명하는 예측 모형을 만들 수 있게 됩니다. 하지만 이러한 방식은 bias는 상당히 줄일 수 있어도, 과적합이 일어날 수도 있다는 단점이 있습니다.

03 Gradient Boosting

장점

1. 구현이 쉽다.

2. 정확도가 좋다.

3. 굉장히 유연하다.

잔차를 계속 줄여나가는 방식으로 학습하기 때문에 정확도가 좋고, 여러가지 손실함수를 적용할 수 있다.

단점

1. 과적합 발생 가능성이 크다.

2. 메모리 문제 존재

3. 해석이 어렵다.

noise가 발생하면 과적합이 발생하며 반복 수가 많아지면 많은 메모리를 사용해야하고, 각 입력변수에 대한 출력변수가 어떻게 변하는지에 대한 해석이 어렵다.

04 코드 따라하기

1.

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(gbrt.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
Training set accuracy: 1.000
Test set accuracy: 0.958
```

04 코드 따라하기

2.

```
gbrt = GradientBoostingClassifier(learning_rate=0.01, random_state=0)
gbrt.fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(gbrt.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
Training set accuracy: 0.995
Test set accuracy: 0.944
```

04 코드 따라하기

3.

```
gbrt = GradientBoostingClassifier(learning_rate=1,random_state=0)
gbrt.fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(gbrt.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(gbrt.score(X_test, y_test)))
#learning_rate는 너무 작거나 너무 크면 성능이 오히려 떨어진다 -> 조정 신중히
```

```
Training set accuracy: 1.000
Test set accuracy: 0.944
```

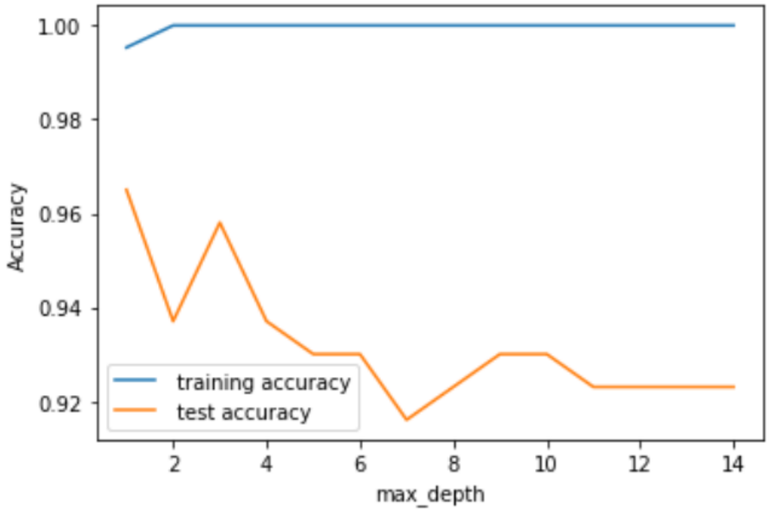
04 코드 따라하기

4.

```
training_accuracy = []
test_accuracy = []

depth_settings = range(1, 15)
for depth in depth_settings:
    gbrt = GradientBoostingClassifier(max_depth = depth)
    gbrt.fit( X_train, y_train)
    training_accuracy.append(gbrt.score( X_train, y_train))
    test_accuracy.append(gbrt.score( X_test, y_test))

plt.plot(depth_settings, training_accuracy, label =" training accuracy")
plt.plot(depth_settings, test_accuracy, label =" test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("max_depth")
plt.legend()
```



04 코드 따라하기

5.

```
gbrt = GradientBoostingClassifier(max_depth=1, random_state=0)
gbrt.fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(gbrt.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
Training set accuracy: 0.995
Test set accuracy: 0.965
```

05 예제 풀어보기

1) 분류문제

붓꽃데이터를 이용한 분류/ 2개의 라벨 고려

```
from sklearn.datasets import load_iris
from sklearn.ensemble import GradientBoostingClassifier

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = [iris.target_names[x] for x in iris.target]
df = df[df['species'].isin(['setosa', 'versicolor'])]
df['species'] = df['species'].map(lambda x: 1 if x == 'setosa' else 0)
df = df.reset_index(drop=True)
```

❺ 예제 풀어보기

반복수 20, 개별 트리 최대깊이는 2, 학습률은 0.5로 설정

Gradient Boosting알고리즘으로 돌리고 학습데이터에 대한 정확도 계산

```
%%time
X = df[['petal length (cm)', 'petal width (cm)']]
y = df['species']
clf = GradientTreeBoosting()
clf.fit(X, y, M=20, max_depth=2,learning_rate=0.5, type_of_col=['continuous']*2)

print('정확도 : ', np.mean(y == clf.predict(X)))
```

```
정확도 :  1.0
Wall time: 2.61 s
```

SECURIOUS

05 예제 풀어보기

같은 조건으로 sklearn에서 제공하는 Gradient Boosting classifier 사용하기

```
%%time
sk_clf = GradientBoostingClassifier(n_estimators=20,learning_rate=0.5, max_depth=2).fit(X, y)
print('정확도 : ', np.mean(y == sk_clf.predict(X)))
```

```
정확도 :  1.0
Wall time: 11 ms
```

정확도는 같지만 속도가 200배 차이남.

SECURIOUS