머신러닝
(Machine Learning)

LECTURE XIV: 지도 학습 8
(Supervised Learning)

Dai-Gyoung Kim

Department of Applied Mathematics

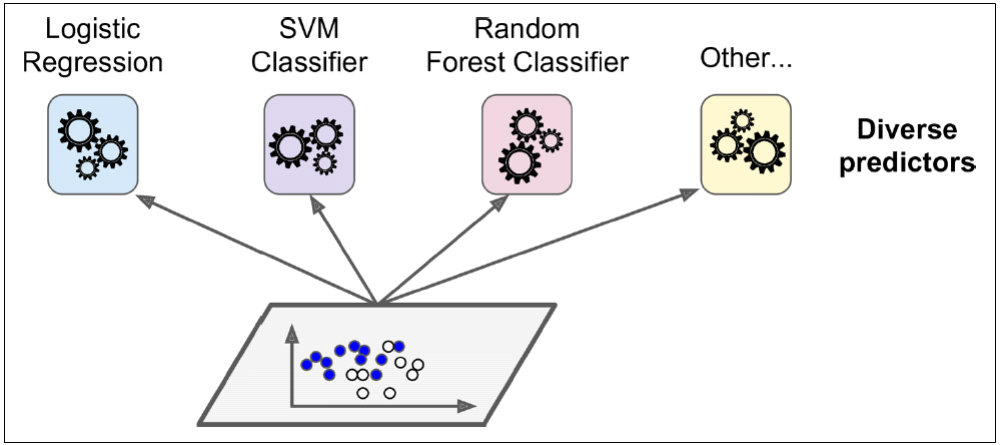
Hanyang University ERICA

지도 학습 (Supervised Learning)

Contents

▶ 결정 트리의 앙상블

일련의 예측기(분류, 회귀 등)로부터 다양한 예측으로부터 최선의 예측을 얻을 수 있음 (대중의 지혜). 이러한 아이디어로부터 일련의 여러 머신러닝 모델을 연결하여 더 강력한 모델을 만드는 기법을 앙상블(ensemble) 학습이라고 함.



- 머신러닝에는 여러 앙상블 모델이 많이 있지만, 그중 다음의 두 앙상블 모델이 분류와 회귀 문제의 다양한 데이터셋에서 효과적이라고 입증되어있음.

  1) **랜덤 포레스트**(Random Forest):

  2) **그래디언트 부스팅**(Gradient Boosting):

  ✓ 위의 앙상블은 모두 결정 트리를 기반으로 하고 있음.

❖ 랜덤 포레스트

- 랜덤 포레스트는 조금씩 다른 여러 결정 트리를 만들어 그 결과를 평균 내어 예측함.
  ✓ 각 트리는 비교적 예측을 잘 할 수 있지만 데이터 일부에 과대적합하는 경향이 있으므로, 이러한 성질을 이용해 서로 다른 방향으로 과대적합된 트리 결과를 평균하여 과대적합을 줄이고 트리모델의 예측성능을 유지할 수 있음.
- 랜덤 포레스트를 구축하려면, 다음과 같은 결정 트리를 많이 만들어야함.
  ✓ 각 트리는 타깃 예측을 잘 해야 함.
  ✓ 각 트리는 다른 트리와 구별되어야 함 (각 트리들이 랜덤하게 생성되도록 함).

- 랜덤 포레스트에서 트리를 랜덤하게 만드는 두 가지 방법.
  1) 트리를 만들 때 사용하는 데이터를 무작위로 선택함(bagging, pasting).
  2) 분할 테스트에서 특성을 무작위로 선택함.

❖ 랜덤 포레스트 구축

1) 랜덤 포레스트를 만들려면 생성할 트리 개수를 정해야함.
   - ✓ 'RandomForestRegressor'나 'RandomForestClassifier'의 매개변수 'n_estimators'.
   - ✓ 여기에서는 트리가 10개가 필요하다고 가정함 (n_estimators=10).

2) 트리를 만들기 전에 데이터의 bootstrap 샘플을 생성함.
   - ✓ 훈련 세트에서 **중복을 허용하여** 무작위로 샘플링함. 이러한 방식을 **배깅**(**bagging**: bootstrap aggregating)이라 함.
   - ✓ 'n_samples'개의 데이터 포인트 중에서 무작위로 'n_samples' 횟수만큼 반복 추출함.
   - ✓ 이 경우 어떤 샘플은 여러 번 중복 추출될 수 있으며, 또한 어떤 샘플은 누락될 수 있음.

- ✓ 어떤 샘플은 한 예측기를 위해 여러 번 샘플링이 되고 어떤 것은 전혀 선택되지 않을 수 있음. 평균적으로 각 예측기에 훈련 샘플의 63% 정도만 샘플링됨.

  ▷ m개의 샘플에서 무작위로 하나를 추출할 때 선택되지 않을 확률은 $1-1/m$이고, 이를 m번 반복했을 때도 선택되지 않을 확률은 $(1-1/m)^m$임.

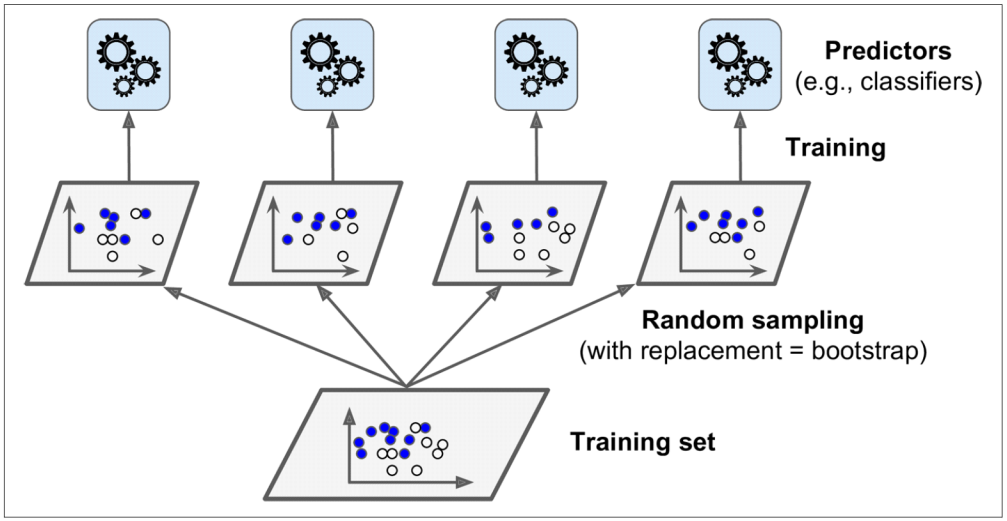
  ▷ $m \to \infty$일 때, $\left(1-\frac{1}{m}\right)^m \to e^{-1}$이므로 샘플링 확률은 $1-e^{-1}=0.63212$임.

- ✓ 선택되지 않은 훈련 샘플의 나머지 37%를 **oob**(out-of-bag) 샘플이라고 함. 각 예측기마다 남겨진 37%는 모두 다름.
- ✓ 예측기가 훈련되는 동안에는 oob 샘플을 사용하지 않으므로 검증 세트나 교차 검증을 사용하지 않고 oob 샘플을 테스트 데이터로 사용해 평가할 수 있음.
- ✓ 앙상블의 평가는 각 예측기의 oob 평가를 평균하여 얻음.

3) bootstrap 샘플링으로 만든 데이터셋으로 결정 트리를 만듦.

- ✓ 이때, 각 노드에서 전체 특성을 대상으로 최선의 테스트를 찾는 것이 아니라, 알고리즘이 각 노드에서 후보 특성을 무작위로 선택한 후 이 후보들 중에서 최선의 테스트를 찾음.
- ✓ 후보 특성의 개수는 'max_feature' 매개변수로 조정함.

- bootstrap 샘플링은 랜덤 포레스트의 트리가 조금씩 다른 데이터셋을 이용해 만들어지며, 또한 각 노드에서 특성의 일부만 사용하기 때문에 트리의 각 분기는 각기 다른 특성 부분 집합을 사용함.
  - ✓ 위의 두 메커니즘이 합쳐져서 랜덤 포레스트의 모든 트리가 서로 달라지도록 만듦.



- 랜덤 포레스트에서 핵심 매개변수는 'max_features'임.
  - ✓ 'max_features = n_features'로 설정: 트리의 각 분기에서 모든 특성을 고려함.
  - ✓ 'max_features = 1'로 설정: 트리의 각 분기에서 무작위로 하나의 특성만을 선택함.
  - ✓ 'max_features' 값을 크게 하면, 랜덤 포레스트의 트리들은 매우 비슷해지고 가장 두드러진 특성을 이용해 데이터에 잘 맞춰짐.
  - ✓ 'max_features' 값을 작게 하면, 랜덤 포레스트의 트리들은 다양해지고 각 트리는 데이터에 맞추기 위해 깊이가 깊어지게 됨.

- 랜덤 포레스트로 예측을 할 때는 먼저 모델에 있는 모든 트리의 예측을 만들고 모든 예측들을 모아 최종 예측을 만듦.
  - ✓ 회귀의 경우 이 예측들을 평균하여 최종 예측을 만듦.
  - ✓ 분류의 경우 **약한 투표 전략**을 사용함. 각 알고리즘이 가능성 있는 출력 레이블의 확률을 제공함으로써 간접적인 예측을 함 (트리들이 예측한 확률을 평균하여 가장 높은 확률을 가진 클래스가 최종 예측값이 됨).

❖ 랜덤 포레스트 분석

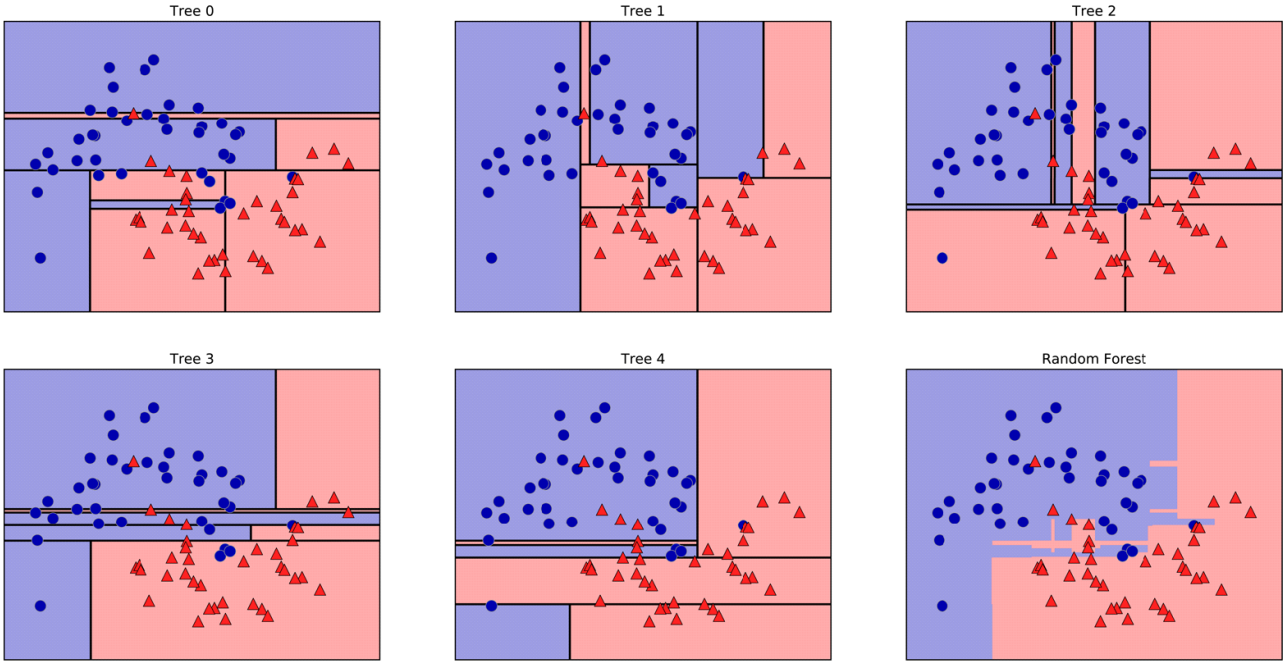
- ‘two_moon’ 데이터셋을 사용하여 트리 5개로 구성된 랜덤 포레스트 모델을 만들어 분석하고자 함.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42)

forest = RandomForestClassifier(n_estimators=5, random_state=2)
forest.fit(X_train, y_train)
```

✓ 랜덤 포레스트 안에 만들어진 트리는 ‘estimator_’ 속성에 저장됨.

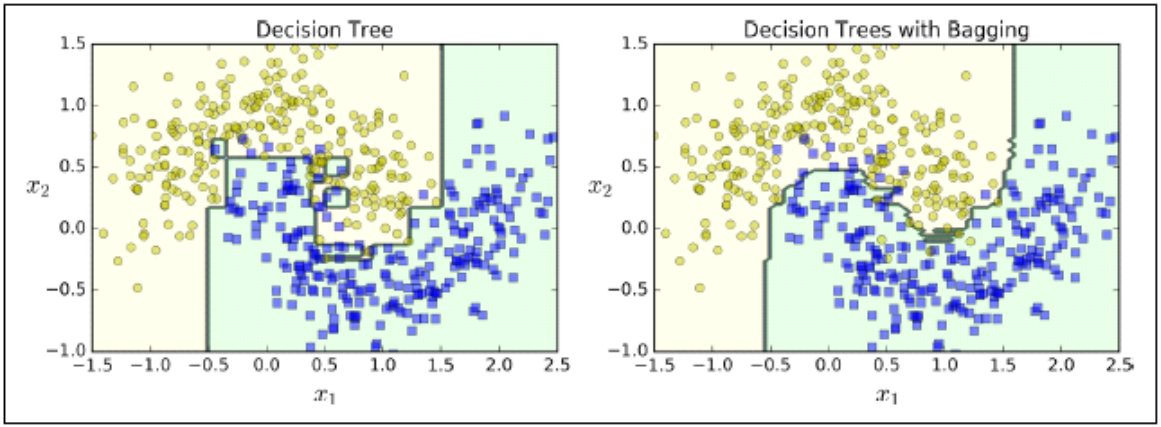
- 다음은 각 트리에서 학습된 결정 경계와 이를 취합한 결정 경계를 시각화하는 코드임.

```
fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for i, (ax, tree) in enumerate(zip(axes.ravel(), forest.estimators_)):
    ax.set_title("Tree {}".format(i))
    mglearn.plots.plot_tree_partition(X_train, y_train, tree, ax=ax)

mglearn.plots.plot_2d_separator(forest, X_train, fill=True, ax=axes[-1, -1], alpha=.4)
axes[-1, -1].set_title("Random Forest")
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
```



✓ 다섯 개의 무작위 결정 트리의 결정 경계와 예측한 확률을 평균 내어 만든 결정경계임.

- 랜덤 포레스트는 개개의 트리보다는 덜 과대적합되고 훨씬 좋은 경계를 만듦.
- 다음 그림은 'two_moons' 데이터셋에 훈련시킨 단일 결정 트리의 결정 경계와 500개의 트리를 사용한 배깅 앙상블의 결정 경계를 비교한 것임.



✓ 앙상블 예측이 결정 트리 하나의 예측보다 일반화가 잘됨.

- 다음 예시에서는 유방암 데이터셋에 100개의 트리로 이루어진 랜덤 포레스트를 적용.

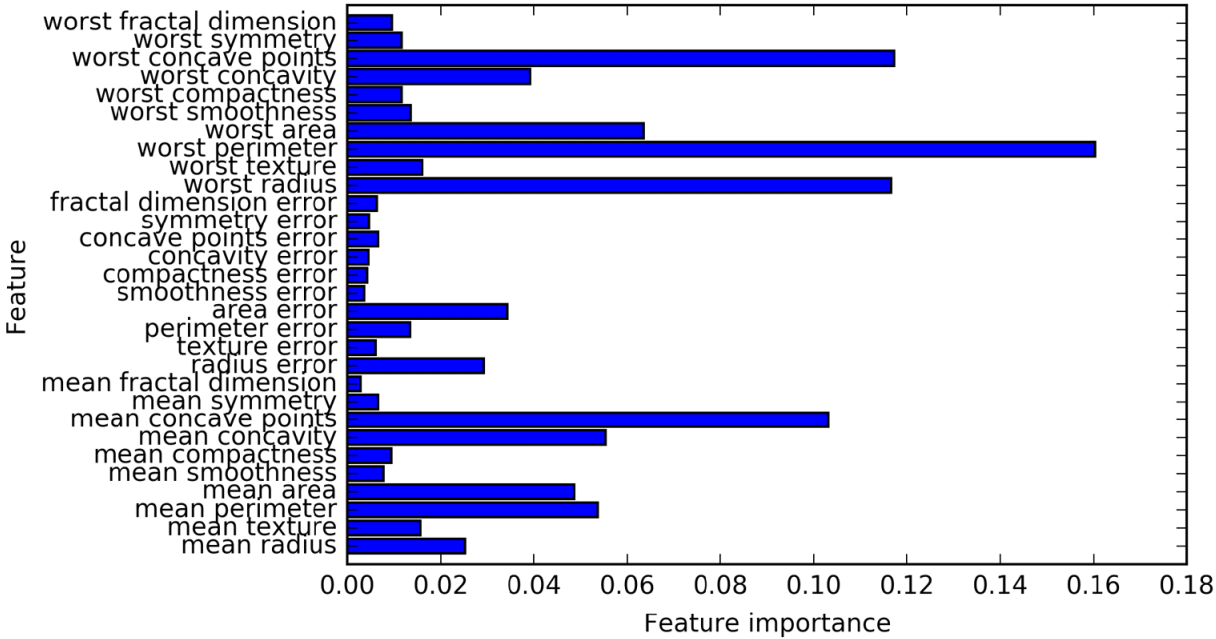
```
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=0)
forest = RandomForestClassifier(n_estimators=100, random_state=0)
forest.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(forest.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.972
```

✓ 랜덤 포레스트는 아무런 매개변수 튜닝 없이도 선형 모델이나 단일 결정 트리보다 높은 97%의 정확도를 냄.

- 결정 트리처럼 랜덤 포레스트도 특성의 중요도를 제공함. 각 트리의 중요도를 취합하여 계산함.
- 일반적으로 랜덤 포레스트에서 제공하는 특성 중요도가 하나의 트리에서 제공하는 것보다 더 신뢰할 만함.

```
plot_feature_importances_cancer(forest)
```



✓ 위의 랜덤 포레스트에서는 'worst radius', 'worst concave points', 'worst perimeter' 특성이 매우 중요함. 가장 많은 정보를 가진 특성으로는 'worst perimeter'를 선택함.

- 랜덤 포레스트를 만드는 무작위성은 알고리즘이 가능성 있는 많은 경우를 고려할 수 있게 하므로, 랜덤 포레스트가 단일 트리보다 더 넓은 시각으로 데이터를 분석할 수 있음.

❖ 장단점과 매개변수

[장단점]

- 회귀와 분류에 있어서 랜덤 포레스트는 현재 가장 널리 사용되는 머신러닝 시스템임.
  - ✓ 성능이 매우 뛰어남.
  - ✓ 매개변수 튜닝을 많이 하지 않아도 잘 작동함.
  - ✓ 데이터의 스케일을 맞출 필요가 없음.

- 랜덤 포레스트는 단일 트리의 단점을 보완하고 장점은 그대로 갖고 있음. 그러나 의사 결정 과정을 간소하게 표현하려면 단일 트리를 사용함. 특히 비전문가에게 예측 과정을 시각적으로 보여주려면 하나의 결정 트리가 더 좋음.

- 랜덤 포레스트 모델을 만들 때 다소 시간이 걸림(병렬처리로 이를 극복할 수 있음).
  - ✓ 멀티 코어 프로세서일 때 'n_jobs' 매개변수를 이용하여 사용할 코어 수를 지정할 수 있음.
  - ✓ 'n_jobs'의 기본값은 1임. 'n_jobs=-1'로 지정하면 컴퓨터의 모든 코어를 사용함.
- 랜덤 포레스트는 텍스트 데이터 같이 매우 차원이 높고 희소한 데이터에는 잘 작동하지 않음 (이런 데이터에는 선형 모델이 더 적합함).
- 랜덤 포레스트는 매우 큰 데이터셋에도 잘 작동하지만, 선형 모델보다 많은 메모리를 사용하여 훈련과 예측이 느림 (속도와 메모리 사용에 제약이 있는 문제라면 선형 모델이 더 적합할 수 있음).

[매개변수]

- 랜덤 포레스트의 중요 매개 변수는 'n_estimators', 'max_features', 'max_depth'임.
  - ✓ 'max_features'는 각 트리가 얼마나 무작위가 될지 결정하여, 작은 'max_features'는 과대적합을 줄여줌. 분류에서는 'max_features=sqrt(n_features)'이고, 회귀에서는 'max_features=n_features'임.

❖ 그래디언트 부스팅 회귀 트리

- 그래디언트 부스팅(Gradient Boosting) 회귀 트리는 여러 개의 결정 트리를 묶어 강력한 모델을 만드는 앙상블임.
  - ✓ 그래디언트 부스팅의 기본적인 아이디어는 트리의 깊이를 낮게(1~5정도의 깊이)하여 약한 학습기를 많이 만들어 연결하여 성능을 향상시킴.
  - ✓ 이름은 회귀지만 이 모델은 회귀와 분류 모두에 사용될 수 있음.

- 그래디언트 부스팅(Gradient Boosting)은 앙상블에 이전까지의 오차를 보완하는 방식으로 예측기(트리)를 순차적으로 추가하여 이전 예측기가 만든 **잔여오차** (residual error)에 새로운 예측기를 학습시킴.

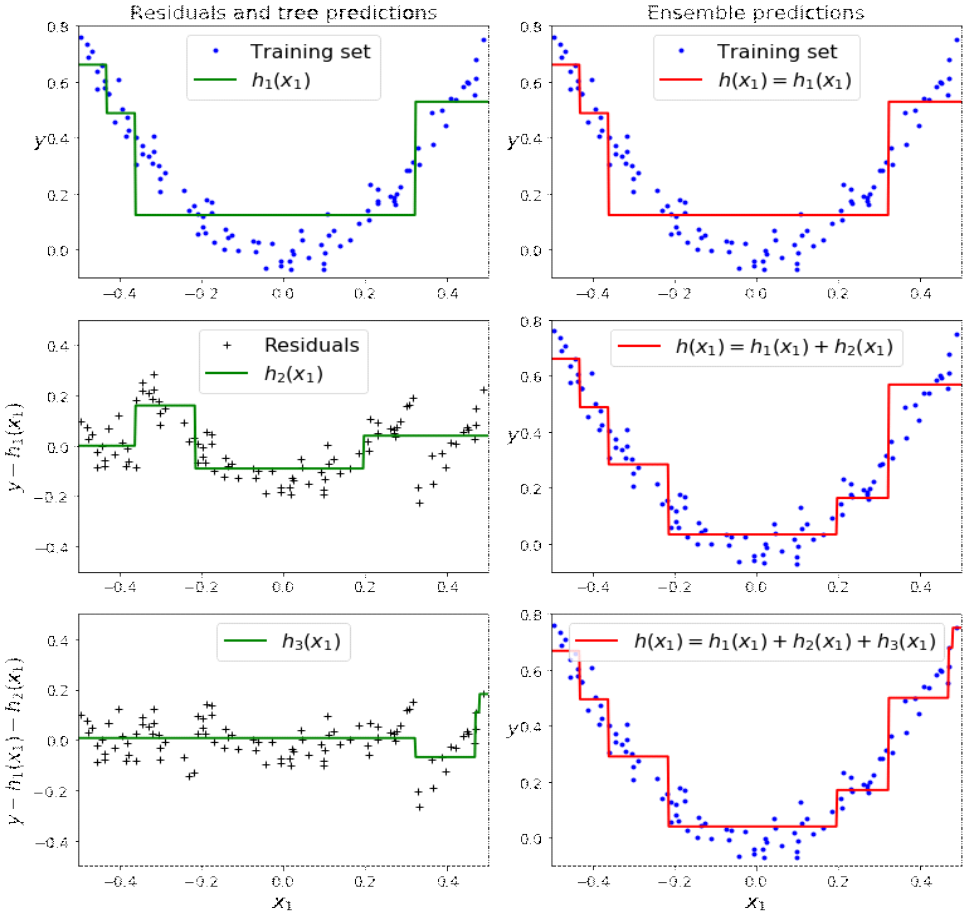
- 기본적으로 그래디언트 부스팅 회귀 트리에는 무작위성이 없고, 대신에 강력한 사전 가지치기가 사용됨.

- 그래디언트 부스팅에서 중요한 매개변수는 이전 트리의 오차를 얼마나 강력하게 보정할 것인지를 제어하는 학습률('learning_rate')임.
  - ✓ 'learning_rate'를 크게 하면 트리는 강하게 보정하기 때문에 복잡한 모델을 만듦.
  - ✓ 'n_estimator'를 키우면 앙상블에 트리가 더 많이 추가되어 모델의 복잡도가 커지고 훈련 세트에서의 오차를 보정할 수 있음.

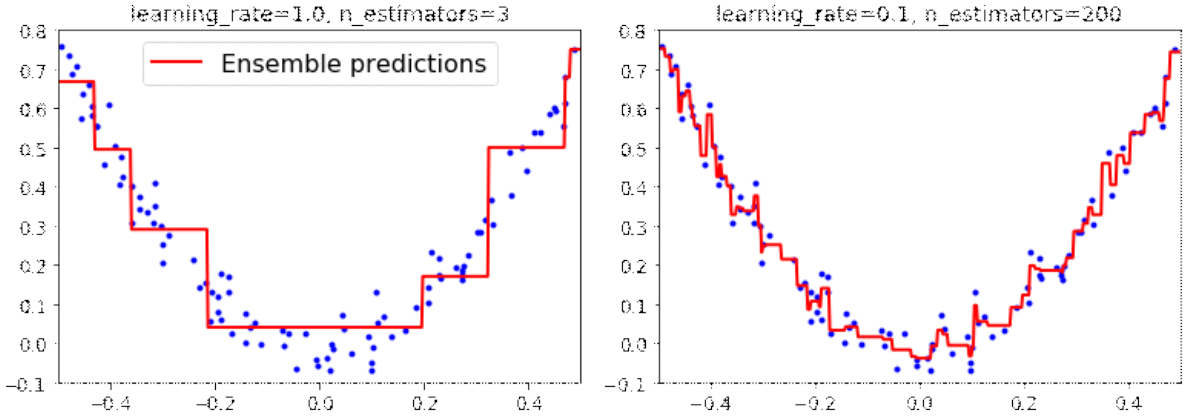
- 그래디언트 부스팅 트리는 머신러닝 분야에서 좋은 성능을 보여주고 있으며 산업계에서도 널리 사용됨.

- 다음 그림의 왼쪽 열은 세 트리의 예측이고 오른쪽 열은 앙상블의 예측임.
- 사이킷런의 'GradientBoostingRegressor'를 사용하면 그래디언트 부스팅 회귀 트리 앙상블을 간단하게 훈련시킬 수 있음.

```
from sklearn.ensemble import GradientBoostingRegressor
gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0)
gbrt.fit(X, y)
```


Residuals and tree predictions
Ensemble predictions
Training set
$h_1(x_1)$
$h(x_1) = h_1(x_1)$
Residuals
$h_2(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1)$
$h_3(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$
y
$y - h_1(x_1)$
$y - h_1(x_1) - h_2(x_1)$
x_1

- 다음 그림은 작은 학습률로 훈련시킨 두 개의 그래디언트 부스팅 회귀 트리 앙상블을 보여줌.



✓ 왼쪽은 훈련 세트를 학습하기에는 트리가 충분하지 않음, 반면 오른쪽은 트리가 너무 많아 훈련 세트에 과대적합되어 있음.

- 다음은 유방암 데이터셋에 'GradientBoostingClassifier'를 학습시킨 코드임.
  - ✓ 기본값은 트리 깊이가 3, 트리 개수가 100, 학습률이 0.1임.

```
from sklearn.ensemble import GradientBoostingClassifier

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=0)
gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(gbrt.score(X_train, y_train)))
print("Accuracy on test set: {:.3f)".format(gbrt.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.958
```

  - ✓ 훈련 세트의 정확도가 100%인 것은 이 모델이 과대적합된 것으로 보임. 과대적합을 줄이기 위해서는 사전 가지치기를 하거나 학습률을 낮추어야함.

- 다음은 과대적합을 줄여 일반화 성능을 높은 결과임.

```
gbrt = GradientBoostingClassifier(random_state=0, max_depth=1)
gbrt.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(gbrt.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt.score(X_test, y_test)))
```

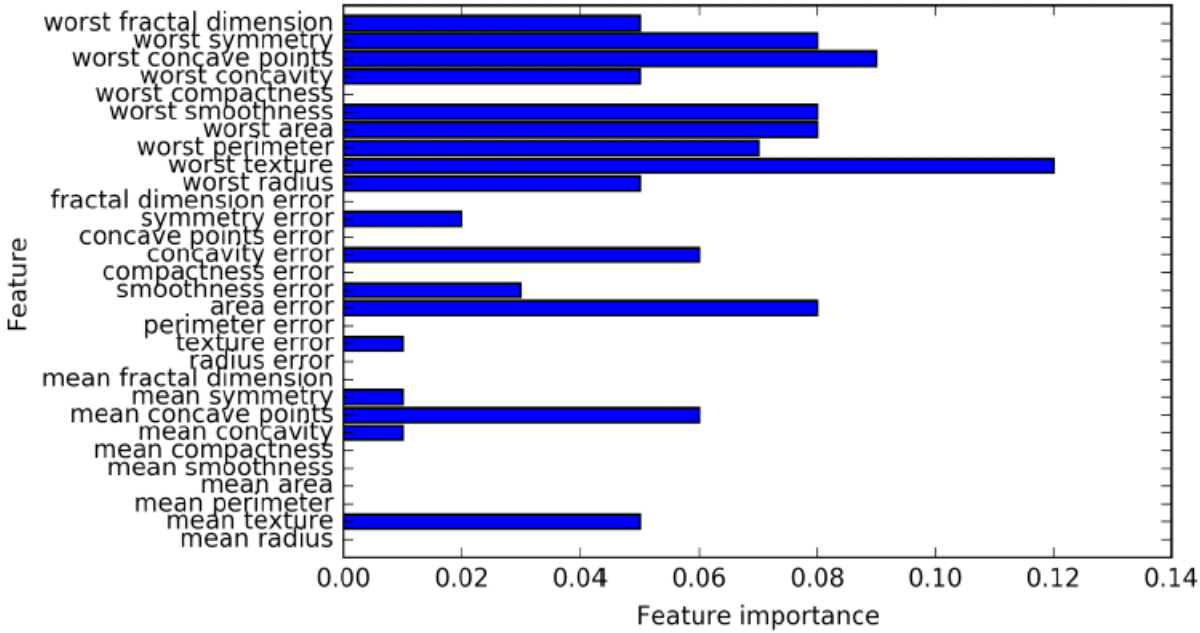
```
Accuracy on training set: 0.991
Accuracy on test set: 0.972
```

```
gbrt = GradientBoostingClassifier(random_state=0, learning_rate=0.01)
gbrt.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(gbrt.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
Accuracy on training set: 0.988
Accuracy on test set: 0.965
```

- 다음과 같이 특성의 중요도를 시각화하면 모델을 더 잘 분석할 수 있음.

```
gbrt = GradientBoostingClassifier(random_state=0, max_depth=1)
gbrt.fit(X_train, y_train)

plot_feature_importances_cancer(gbrt)
```



✓ 그래디언트 부스팅 트리의 특성 중요도도 랜덤 포레스트의 특성 중요도와 비슷함. 단지 랜덤 포레스트의 경우와 달리 일부 특성을 완전히 무시하고 있음.

- 비슷한 종류의 데이터에서는 그래디언트 부스팅이나 랜덤 포레스트 둘 다 잘 작동함.
  - ✓ 랜덤 포레스트는 더 안정적으로 작동함.
  - ✓ 그래디언트 부스팅은 더 예측 시간이 빠름.

❖ 장단점과 매개변수

[장단점]

- 그래디언트 부스팅 결정 트리는 지도 학습에서 가장 강력하고 널리 사용되는 모델 중에 하나임.
  1) 특성 스케일을 조정하지 않아도 됨.
  2) 이진 특성이나 연속적인 특성에서도 잘 작동함.

- 그래디언트 부스팅 결정 트리의 가장 큰 단점은 다음과 같음.

  1) 매개변수를 잘 조정해야함.

  2) 훈련시간이 많이 소요됨.

[매개변수]

- 그래디언트 부스팅 결정 트리의 중요 매개변수는 다음과 같음.

  1) 'n_estimator': 트리의 개수를 조정.

  2) 'learning_rate': 이전 트리의 오차를 보정하는 정도를 조절하는 학습률.

  ✓ 'n_estimator'와 'learning_rate'의 두 매개변수는 밀접한 관계가 있음.
  ('learning_rate'를 낮추면 비슷한 복잡도의 모델을 만들기 위해 더 많은 트리를 추가하여야함)
  ('n_estimator'를 크게 하면 모델이 복잡해지고 과대적합될 가능성이 높음)

  3) 'max_depth', 'max_leaf_nodes': 각 트리의 복잡도를 조절함.

  ✓ 일반적으로 그래디언트 부스팅 모델에서는 'max_depth'를 작게(1~5) 설정함.