Chap 7

Ensemble Learning and Random Forests

Written by Jin Kim

191 Ensemble Learning and Random Forests

- 수천만명에게 어려운 질문을 하고 종합한다고 해 보자. 많은 경우에 한 명의 전문가의 대답보다 종합된 대답이 나을 것이다.
- 마찬가지로 predictor(classifiers or regressors)들의 prediction을 종합한다면 best individual predictor 보다 더 나은 결과를 얻을 수도 있을 것이다.
  - 이 predictor들의 그룹을 *ensemble* 라고 부른다.
  - 따라서, 이 테크닉을 *ensemble learning* 이라고 부른다.
  - Ensemble Learning algorith을 *Ensemble method* 라고 부른다.
- 위 방법은, Decision Tree classifier 를 train 셋의 부분집합에 대해 각각 train 한다고 해 보자. 예측을 하기 위해 각 분류기의 prediction 에서 가장 많은 표를 받은 클래스가 결과값이 되는데, 이것을 *Random Forest* 라 부른다.

191 Ensemble Learning and Random Forests

- 몇 가지의 좋은 predictor들을 만들었다면, 그것들을 조합하여 더 나은 predictor로 만드는게 좋다. 실제로도 여러가지 조합으로 머신러닝 경진대회에서 우승을 하는 경우도 많음.
- 이번 챕터에서는 *bagging*, *boosting*, *stacking* 등을 포함하여 여러가지 Ensemble method 들에 대해 알아볼것이다.
- Random Forests에 대해서도 알아볼것이다.

192 Voting Classifiers

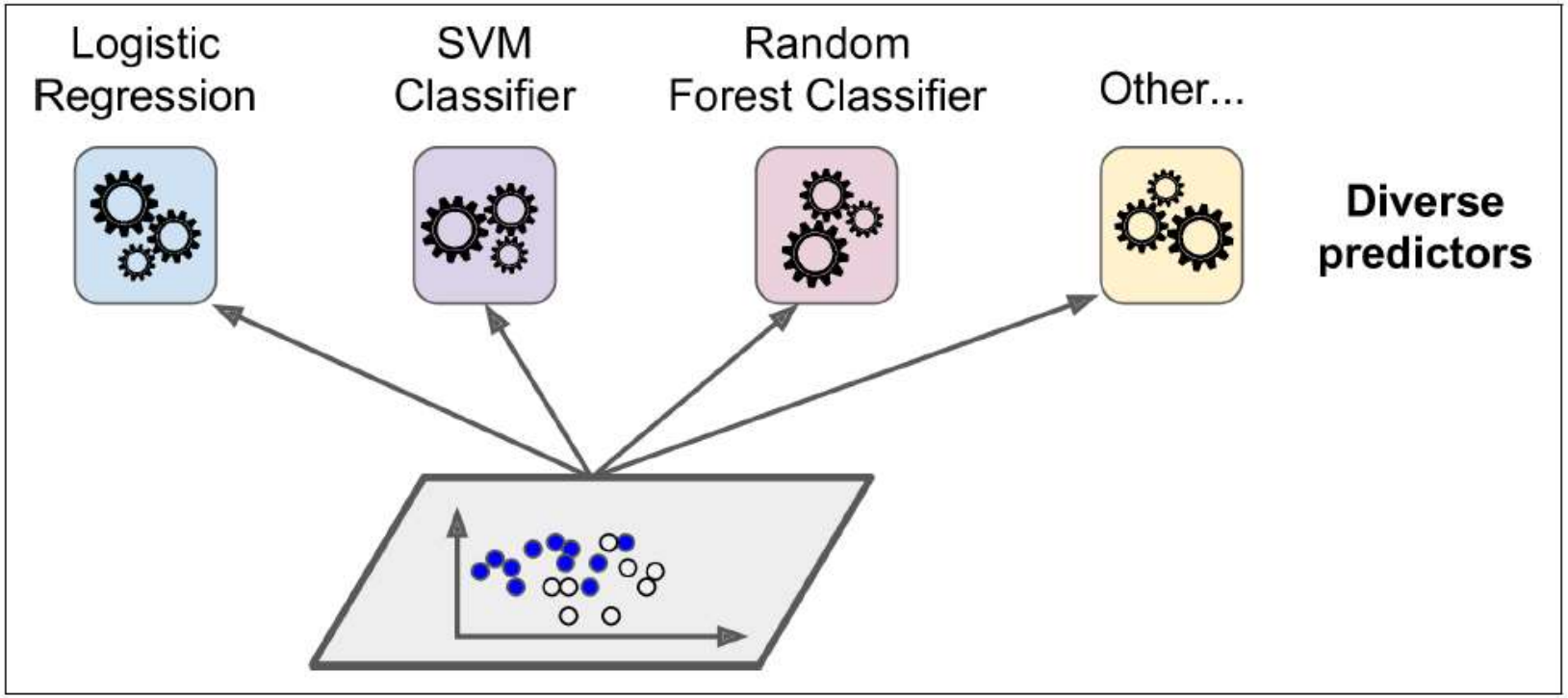

Figure 7-1. Training diverse classifiers

여러가지의 classifier를 가지고 있다고 해 보자.

192~193 Voting Classifiers

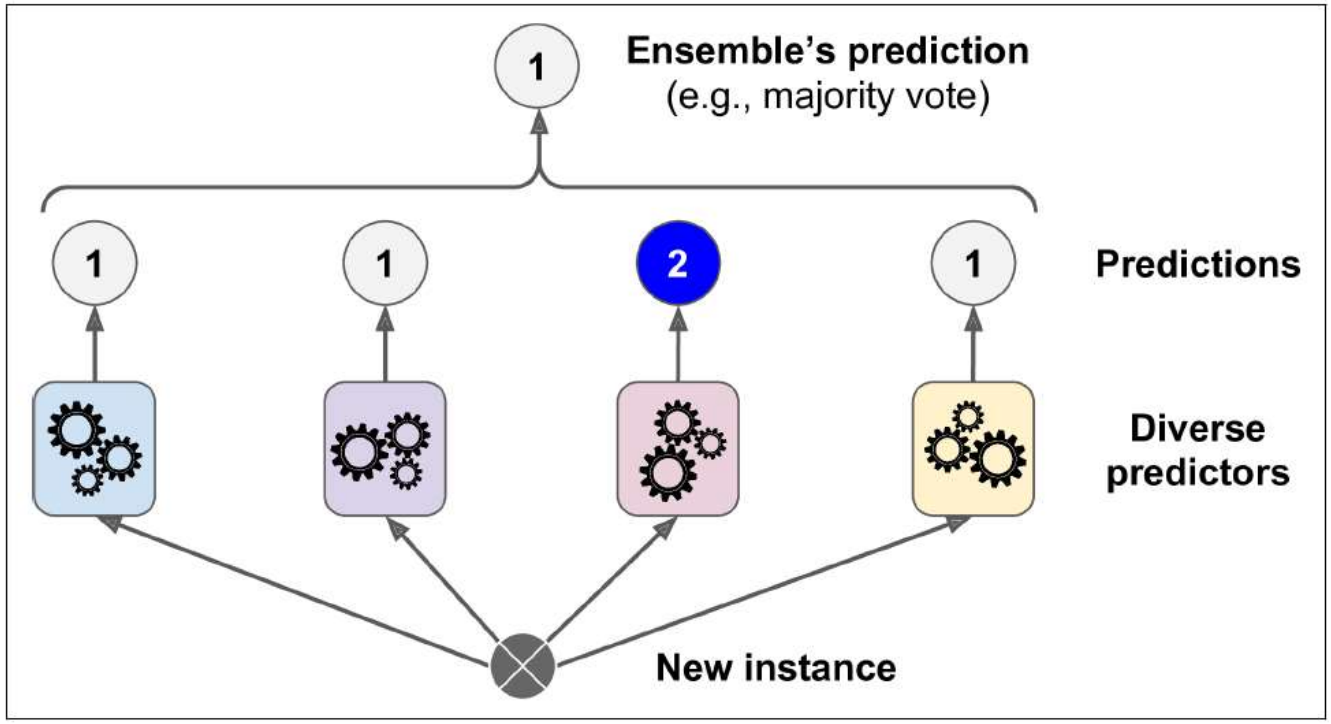

Figure 7-2. Hard voting classifier predictions

더 나은 classifier를 만드는 방법은 prediction들을 종합해서 가장 많은 표를 얻은 class를 predict하는것이다.

이런 방법을 *Hard voting classifier predictions* 라 부른다.

이렇게 해서 만들어진 classifier들은 해당 앙상블에서의 best classifier보다 더 높은 정확성을 가질 경우가 많다.
weak learner(굉장히 정확도가 낮은 classifier)들을 앙상블하여도 *strong learner*로 만들 수 있다. -> 이 경우 많은 weak learne와 충분한 다양성이 있어야함.

193 Voting Classifiers

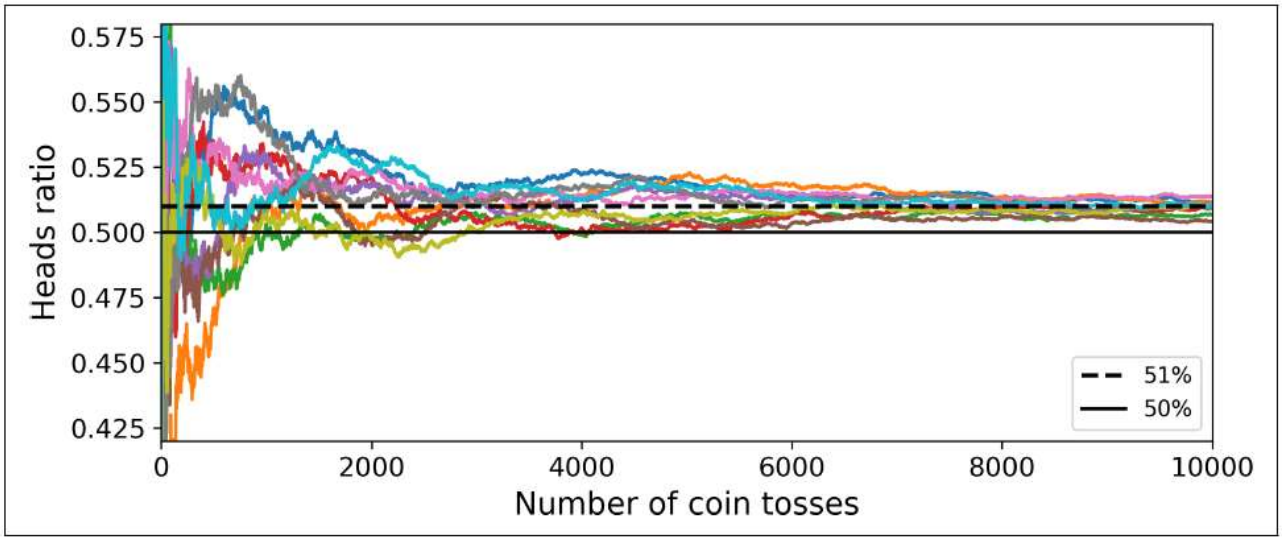

Figure 7-3. The law of large numbers

각각이 51%의 확률로 정답을 맞추는 1000개의 classifier 를 포함하는 Ensemble 을 만들었다고 해 보자. 이 Ensemble로는 75%의 정확도를 가질 수 있다. 그러나 이것은 모든 classifier가 완벽히 독립되고 uncorrelated(서로 상관없는) 에러를 가지고 있을때만 가능하다. 즉, 서로다른 1000개의 데이터를 사용해야 하는데 위 상황에서는 힘들다. 같은 타입의 에러를 만들것이고, 잘못된 클래스로 투표하는 classifier들도 많을것이다.

51%확률로 앞면, 49%확률로 뒷면이 나오는 코인이 있다고 가정하자.
만약 천번을 던진다면 보통은 510번의 앞면, 490번의 뒷면이 나올것임.
1000번을 던진 후에 앞면이 뒷면보다 더 많이 나올 확률은 75%.
10000번을 던진 후에는 97%임.
이는 *law of large numbers* 라고 부름. -> 코인을 던질수록 확률은 51%에 가까워짐.

좌측 그래프는 10개의 코인던지기 그래프임. 던지는 횟수가 많아질수록 비율은 51%로 감.

194 Voting Classifiers

- Ensemble은 각 predictor들간이 독립적일때 가장 잘 작동한다.
  - 이렇게 독립적으로 만드는 방법은, 각 predictor들이 매우 다른 알고리즘을 사용하는것임.
- Colab

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

Let's look at each classifier's accuracy on the test set:

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

194 Voting Classifiers

- 모든 classifier가 각 가능성을 예측할 수 있다면, 가장 높은 가능성을 가진 class를 예측하게 만든다.
- 이것은 *soft voting* 이라 함.
  - 보통은 hard voting보다는 적중률이 높은데, highly confident vote에 weight를 주기 때문.

195 Bagging(*bootstrap aggregating)* and Pasting

- Classifier들을 다양하게 만드는 방법 중 하나는 앞서 말했듯 다른 알고리즘을 사용하여 predictor를 만드는것임.
- 또하나의 방법은 training set에서 각 predictor마다 다른 sample을 추출하여 만드는것임.
  - 이 과정에서, 샘플 추출시 중복을 허용하냐 안하냐에 따라 bagging과 pasting으로 나뉨.
  - 여러개의 predictor에 대해서는 bagging과 pasting이 여러번 sample 될 수 있지만, 하나의 predictor에 대해서는 bagging만이 여러번 sample 될 수 있음.

195 Bagging(*bootstrap aggregating)* and Pasting

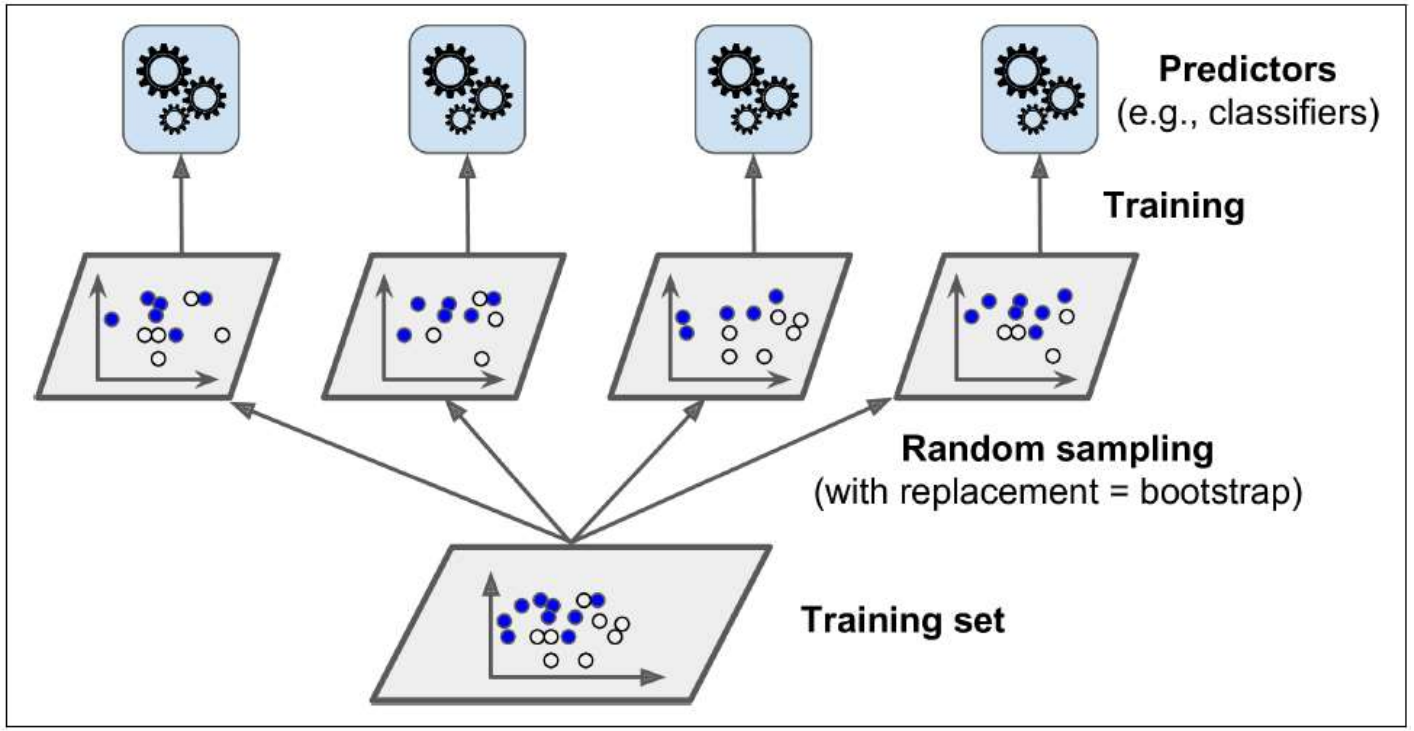

Figure 7-4. Pasting/bagging training set sampling and training

좌측 그림은 위 bagging의 예시임

195~196 Bagging(*bootstrap aggregating)* and Pasting

- 모든 predictor들이 train되면, ensemble을 만들 수 있음.
  - Classifier의 경우 voting으로,
  - Regressor의 경우 값의 average로 할 수 있음.
- 이렇게 ensemble된것은 1개의 predictor에 비해 bias는 비슷하지만, variance 는 줄어듦.
- Predictor들은 CPU 코어나 서버를 각 하나씩 사용하여 병렬작업으로 동시에 train 될 수 있음.

196~197 Bagging and Pasting in Scikit-Learn

- Scikit-learn에서는 BaggingClassifier 또는 BaggingRegressor 의 명령어로 API를 지원함.
- Colab

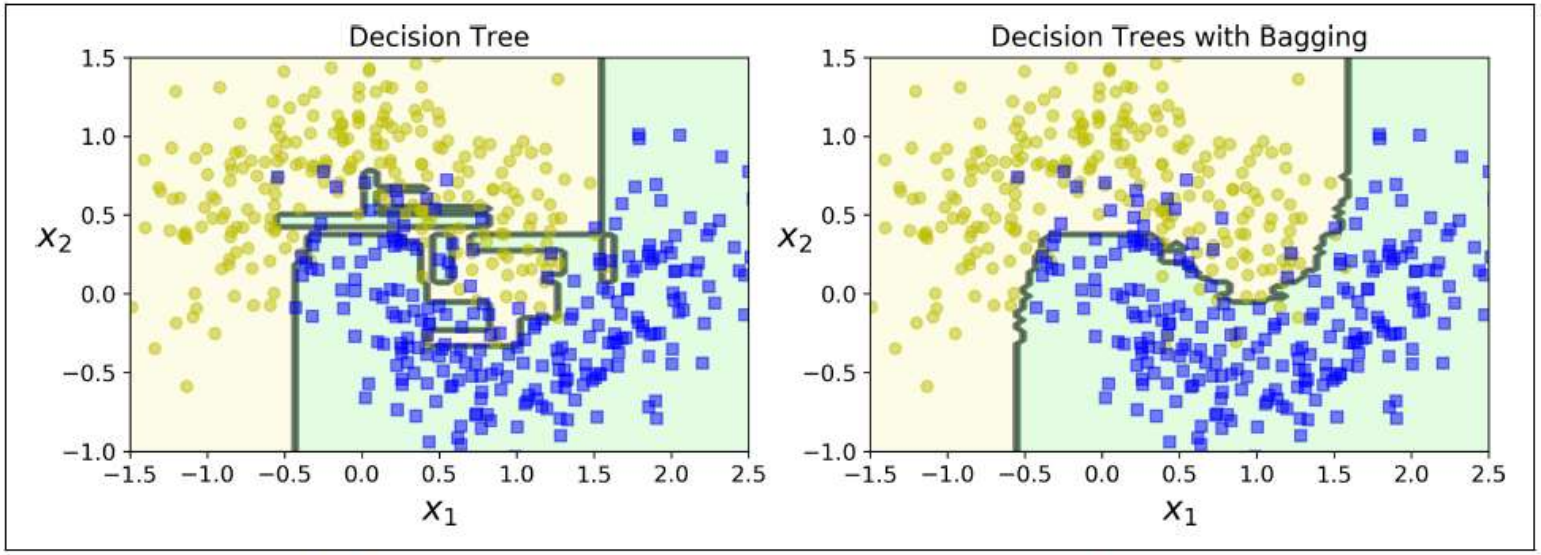

Figure 7-5. A single Decision Tree versus a bagging ensemble of 500 trees

좌측은 일반적인 Decision Tree
우측은 500개 Tree의 bagging ensemble
좌측에 비해 우측은
Bias 는 비슷
Variance는 줄어듦
-> 같은 숫자에 대한 에러는 비슷하게 발생하나, decision boundar는 less irregular함.

197 Bagging vs pasting

- Bootstrapping을 하게되면('With Replacement = WR'—an element may appear multiple times in the one sample) predictor들이 더 다양한 sample을 가지게 된다.
  - Bagging이 pasting에 비해 bias가 살짝 더 높음.
  - Predictor들이 상호간에 less correlated 됨.
  - Ensemble의 variance 가 줄어듦.
- 결론 : 대부분의 경우에 Bagging 이 더 나음.

197~198 Out-of-Bag Evaluation

- Bagging을 하게되면 어떤 instance들은 여러번 샘플 되고, 어떤 instance들은 아예 샘플 되지 않게 됨.
  - 이런 instance들을 *out-of-bag* (oob) instances 라 부름.
  - oob들을 이용해서 validation 처럼 사용할 수 있음.
  - Scikit-learn에서 oob_score=True 를 이용하면 가능
- Colab

198~199 Random Patches and Random Subspaces

- BaggingClassifier는 feature도 sampling 가능.
- high-dimensional inputs 일때 유용하게 사용됨.(image 등)
- Feature와 instance 모두를 sampling 하는것을 *Random Patches* method 라 부른다.
- 모든 instance를 사용하지만, feature만 sampling 하는것을 *Random Subspaces* method 라 부른다.
- Feature를 sampling 하게되면 predictor가 더욱 다양해지지만, bias가 살짝 늘어나고 varianc가 줄어들게 된다.

199 Random Forests

- Decision Trees의 Ensemble된것임.
- RandomForestClassifier 와 RandomForestRegressor 를 사용하면 된다.
- Colab
- Random Forests는 기존 Decision Tree에서 노드를 나눌 때 best feature를 찾는것 대신, random subset of feature 중에서 best feature를 찾음.
  - -> 다양성을 증가시킴. Bias 증가, variance 감소

200 Extra-Trees

- Random Forest에서 각 노드를 정할때 feature의 random subset(무작위 집단)을 기준으로 나눠진다. 근데 best threshold를 찾는것이 아닌 random threshold를 이용하여 이것을 더 랜덤하게 만들 수도 있다.
- 이것을 *Extremely Randomized Trees* ensemble(혹은 *Extra-Trees*) 라 부른다.
- Bias는 올라가고, variance는 줄어든다.
- Scikit-learn의 ExtraTreesClassifier class를 이용해서 만들 수 있다.
- RandomForestClassifier가 나을지, ExtraTreesClassifier가 나을지는 알 수 없다. Cross validation으로 확인하여야만 한다.

200~201 Feature Importance

- Random Forest 는 각 feature가 얼마나 중요한지 쉽게 측정 가능함.
- Scikit-learn은 Feature가 impurity 를 얼마나 줄이는지에 대해 측정함.
  - 노드의 weigh는 feature가 해당 노드에 얼마나 많냐에 따라 정해짐.

- Colab

201~202 Boosting

- 여러가지 약한 learner들을 ensemble하여 강한 learner로 만드는것임.
- 기본적으로는 각 predictor들을 순차적으로 train 하는것인데, 이전의 predictor(predecessor)들을 조정하면서 진행하는것임.
- 현존하는 가장 유명한 boosting은 *AdaBoost(Adaptive Boosting)* 와 *Gradient Boosting* 임.

202 AdaBoost

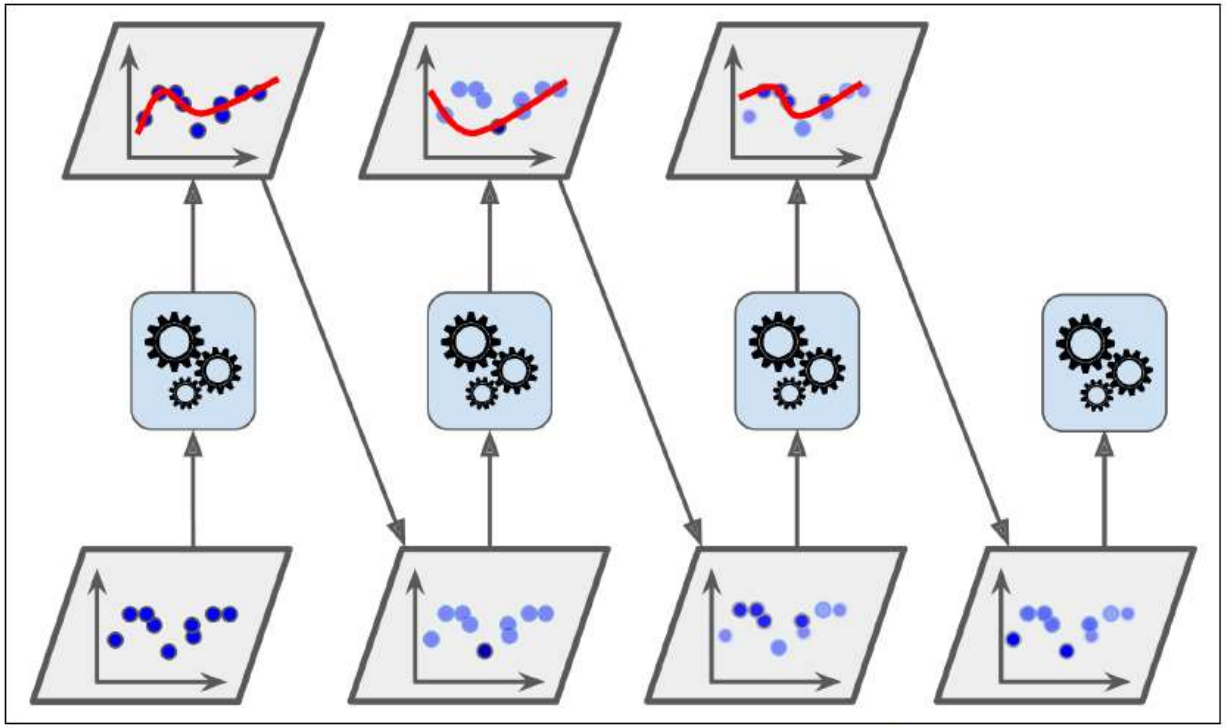

Figure 7-7. AdaBoost sequential training with instance weight updates

Predecessor(이전 predictor)들이 underfit 된 것을 조정하면서 진행하는 방법.

순서

1. 맨처음의 predictor가 먼저 train된다.
2. 해당 모델로 train셋에 prediction 을 한다.
3. 잘못 분류된 instance들의 weigh를 증가시켜 다음 predictor에 적용된다.
4. 다음 predictor을 train하여 위의 과정을 반복한다.

202~203 Decision boundaries

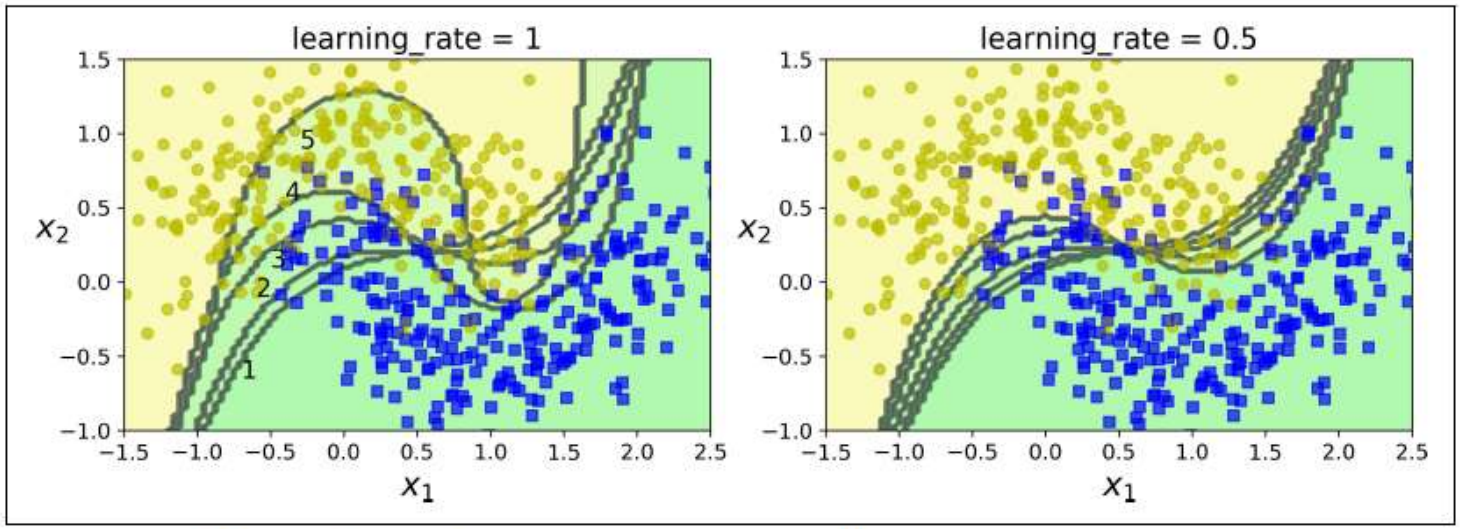

Figure 7-8. Decision boundaries of consecutive predictors

러닝레이트에 따라서
predictor 들의 변화량임.

203~204 AdaBoost

Equation 7-1. Weighted error rate of the j^{th} predictor

$$r_j = \frac{\sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$

where $\hat{y}_j^{(i)}$ is the j^{th} predictor's prediction for the i^{th} instance.

초기 웨이트값은 1/m으로 세팅됨.

Equation 7-2. Predictor weight

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

η는 러닝레이트

Equation 7-3. Weight update rule

for $i = 1, 2, \cdots, m$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp\left(\alpha_j\right) & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

다음 predictor에서도 위 과정을 반복하여 weight 계산함.

204 AdaBoost

Equation 7-4. AdaBoost predictions

$$\hat{y}(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x}) = k}}^{N} \alpha_j \quad \text{where } N \text{ is the number of predictors.}$$

최종적으로 predictor weight αj 를 사용하여 가장 높은 weight vote를 받은 class를 predict함

205 *SAMME*

- 사이킷런에는 *SAMME*(*Stagewise Additive Modeling using a Multiclass Exponential loss function*) 을 사용하는데, AdaBoost의 multiclass 버전임.
- 2개의 클래스밖에 없으면 SAMME와 AdaBoost는 똑같이 동작.
- 3개 이상에서는 probability 를 계산할수있게 동작.
- Colab

205 Gradient Boosting

- AdaBoost와 비슷하게 predictor들을 순차적으로 train하는 Ensemble method임.
- 하지만, weight를 다르게 주는 Adaboost와는 다름.
- 다음 predictor가 이전 predictor의 잔여 오류(실험에 의해 측정한 측정값과 논리적으로 계산한 논리값과의 차) 를 fit함.
  - this method tries to fit the new predictor to the *residual errors* made by the previous predictor.

206 DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

Now train a second `DecisionTreeRegressor` on the residual errors made by the first predictor:

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

잔여오차에 대해 fit함

Then we train a third regressor on the residual errors made by the second predictor:

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

잔여오차에 대해 fit함

Now we have an ensemble containing three trees. It can make predictions on a new instance simply by adding up the predictions of all the trees:

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

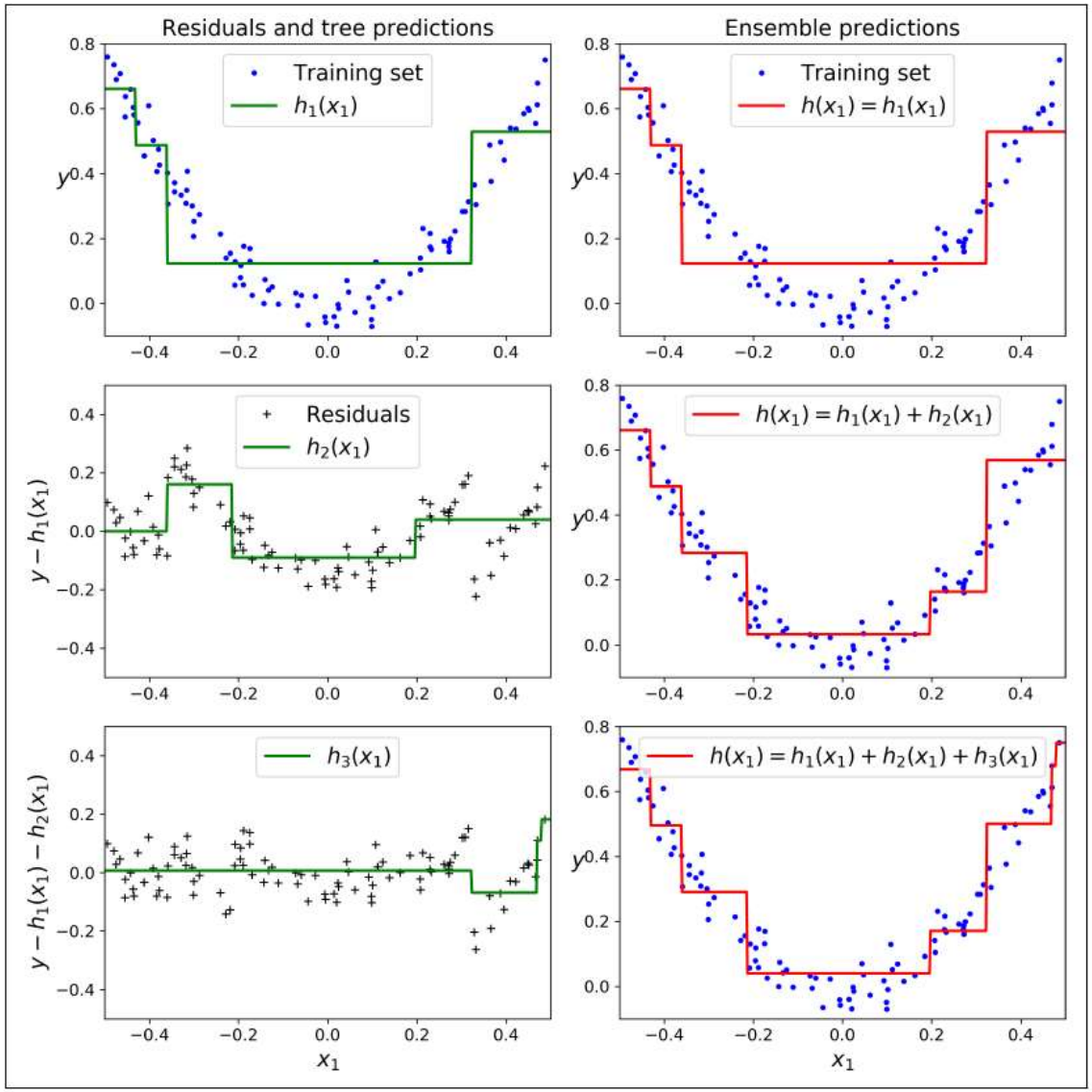

Figure 7-9. Gradient Boosting

207

좌측은 그냥 Decision Tree
우측은 Ensemble로 만들어진 predictor임

맨 윗줄
좌측은 하나의 Decision Tree이며, 우측 역시 하나의 Predictor밖에 없는 Ensemble이기 때문에 결과값이 같음

중간
좌측은 잔여오차에 대한 Decision Tree
우측은 2개의 Decision Tree가 합쳐짐.

아래
잔여오차 / Ensemble

208 Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor

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

좌측의 명령어를 이용하면 위 페이지에서 했던 ensemble을 만들 수 있음.

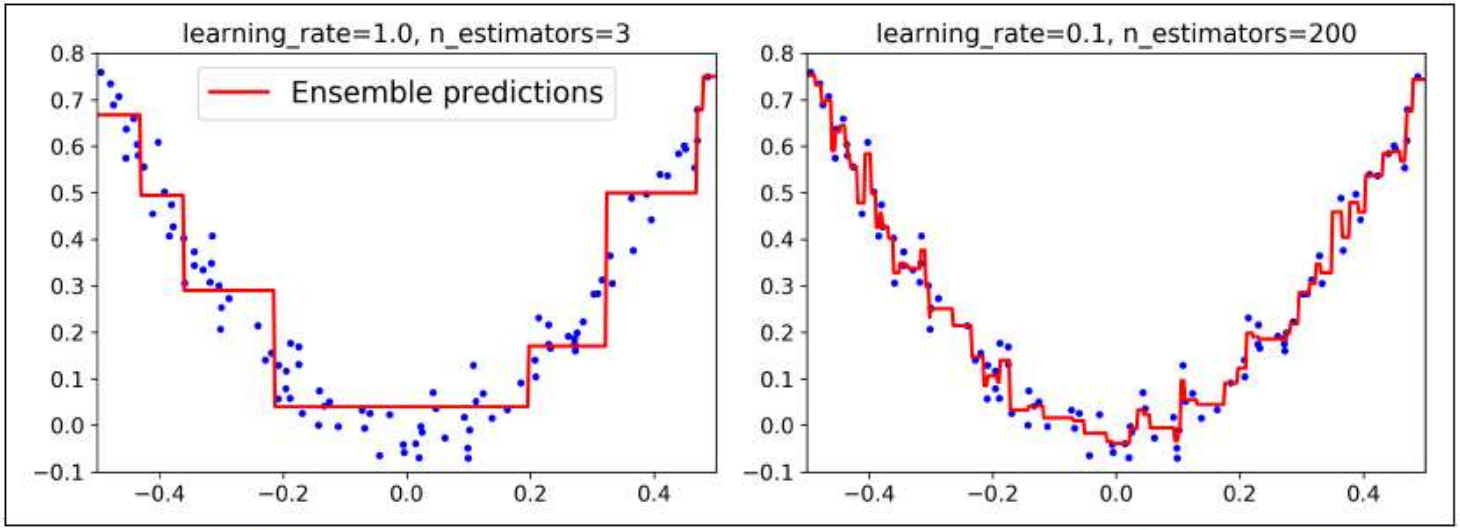

Figure 7-10. GBRT ensembles with not enough predictors (left) and too many (right)

Learning_rate 명령어에 따른 prediction 차이

early stopping을 이용하면 적절한 learning rate를 찾을 수 있음.
혹은 staged_predict() 로도 찾을 수 있음. -> iterator(반복자)를 return하는 함수

209 GBRT ensemble(early stopping)

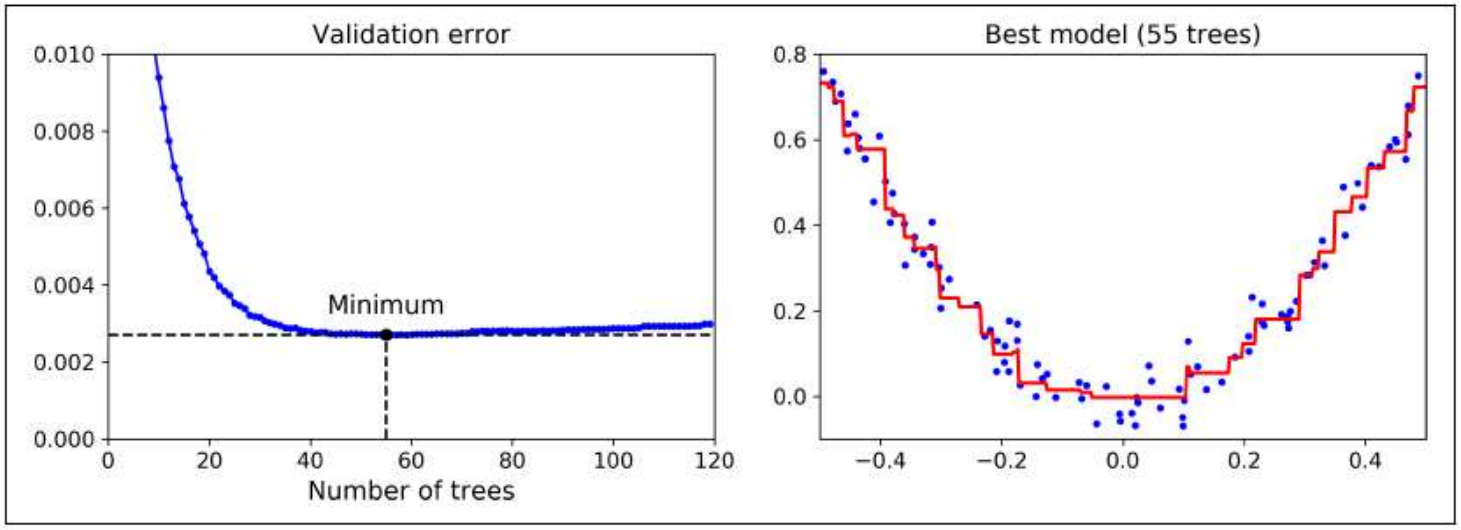

Figure 7-11. Tuning the number of trees using early stopping

좌측 : validation errors
우측 : best model's predictions

209 GradientBoostingRegressor

- Subsample 라는 하이퍼 파라미터도 있음.
- Subsample=0.25라면 각 트리에서 25%만의 랜덤으로 선택된 instanc들만 train되는것임.
  - -> bias 상승, variance 하강
  - train 속도 증가
  - *Stochastic Gradient Boosting* 라고도 불림.

210 *XGBoost*(Extreme Gradient Boosting)

- optimized implementation of Gradient Boosting 을 Scikit-Learn의 라이브러리중 하나인 XGBoost에서 사용가능함.
- extremely fast, scalable and portable이 목적임.

```
import xgboost

xgb_reg = xgboost.XGBRegressor()
xgb_reg.fit(X_train, y_train)
y_pred = xgb_reg.predict(X_val)
```

사용이 매우 쉬움

XGBoost also offers several nice features, such as automatically taking care of early stopping:

```
xgb_reg.fit(X_train, y_train,
            eval_set=[(X_val, y_val)], early_stopping_rounds=2)
y_pred = xgb_reg.predict(X_val)
```

210~211 Stacking(*stacked generalization*)

기존의 방법과는 다르게, model을 train하여 모든 predictor의 prediction 을 합치는것임.

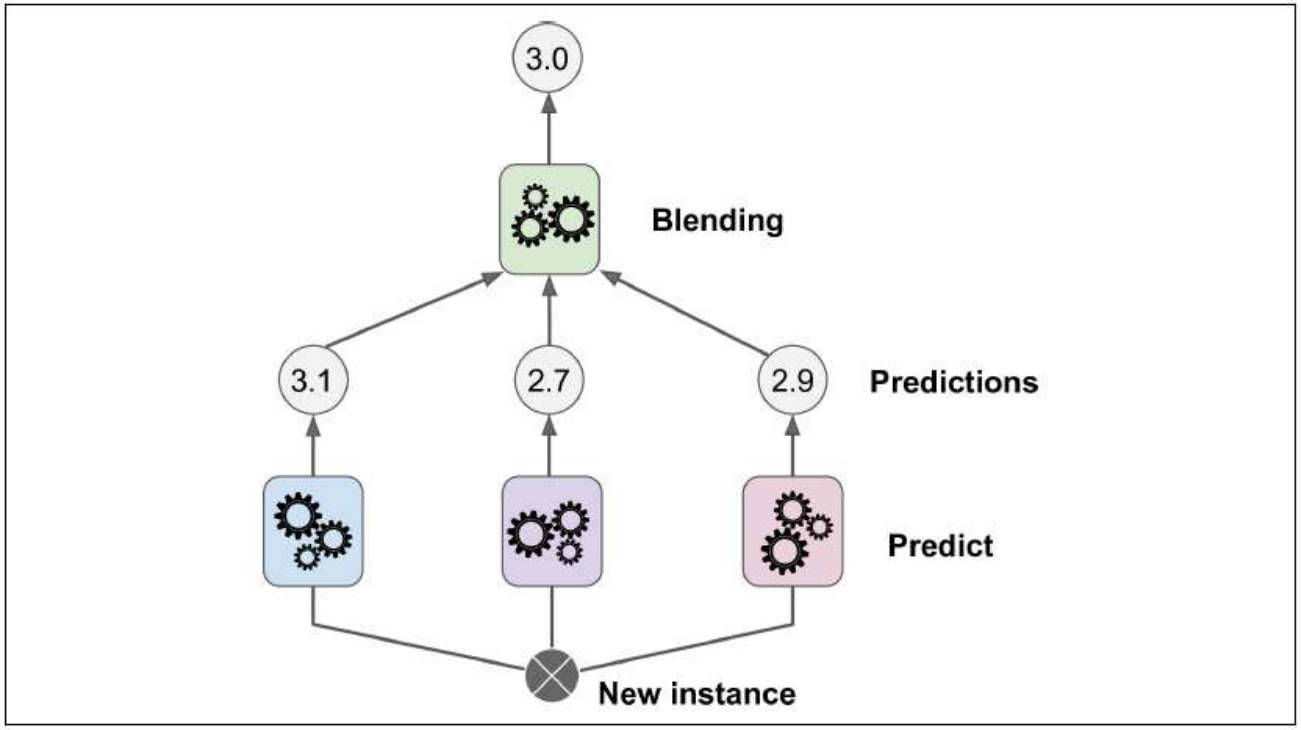

Figure 7-12. Aggregating predictions using a blending predictor

좌측은 위와같은 형태의 regression task임.
아래 각 predictor들이 내놓은 prediction값(3.1등)을 *blender(meta learner)* 는 하나의 input으로 받아들이고, output으로 3.0의 값을 내놓음.

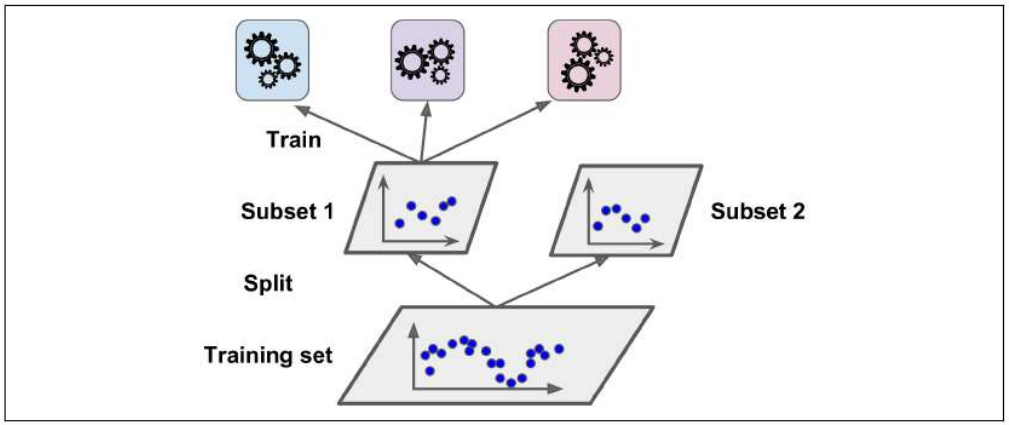

Figure 7-13. Training the first layer

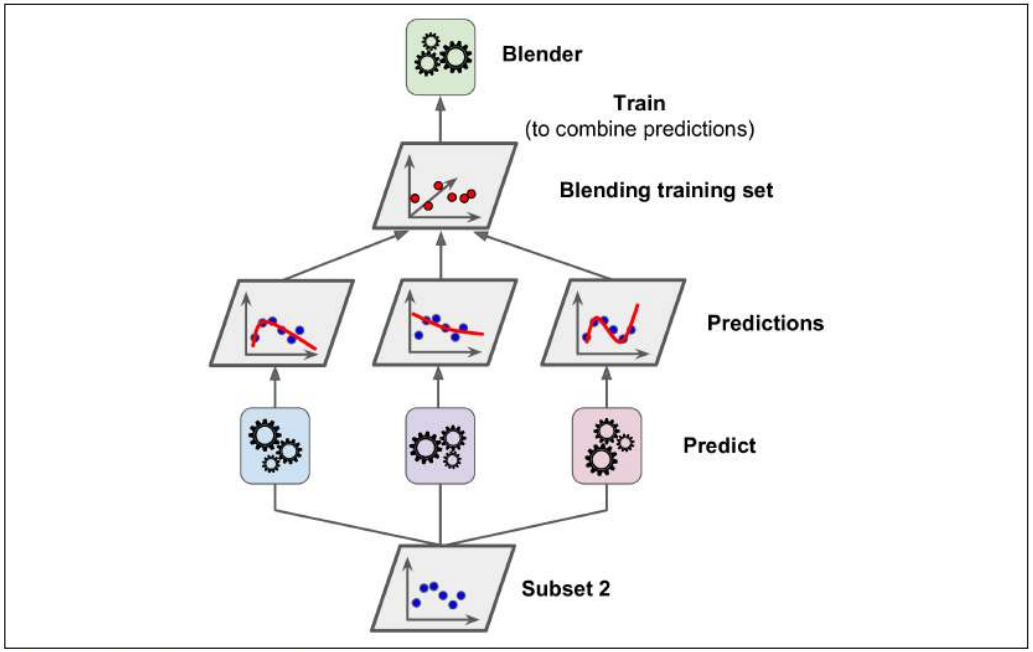

Figure 7-14. Training the blender

211 Blender를 train하기 위한 방법

가장 흔한 방법은 hold-out set 를 사용하는것임.
트레이닝셋을 먼저 둘로 나눈다음, 그중 하나를 이용해 predictor를 train시킨다.
그리고 나머지 한쪽을 방금 만든 predictor를 이용하여 prediction 한다. 이걸 hold-out set 이라고 부름.
2번째 반쪽의 모든 instance들은 3개의 predicted된 값들이 있는데, 이 3개의 값을 input으로 이용해 새로운 training set을 만들 수 있다.

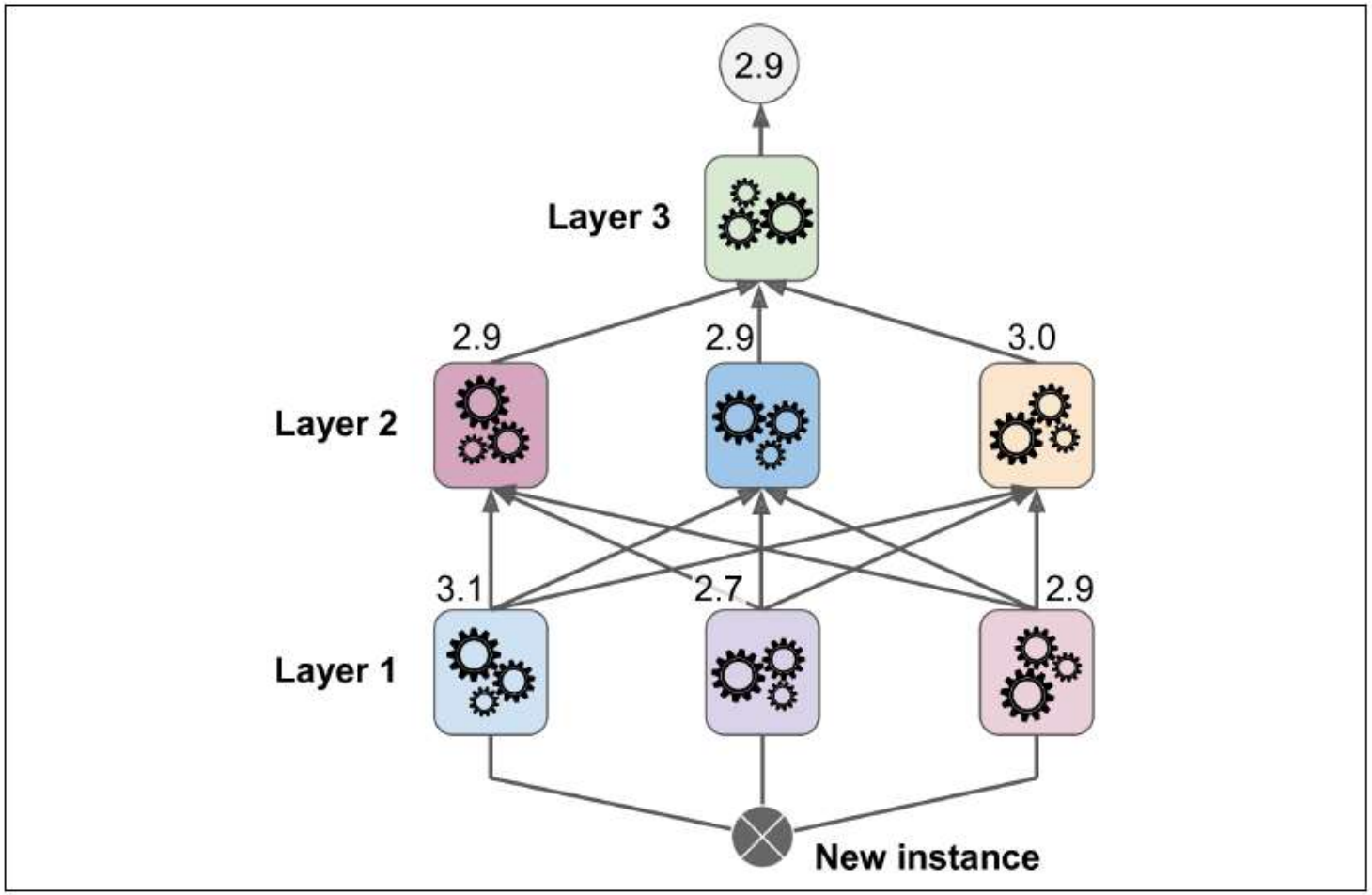

Figure 7-15. Predictions in a multilayer stacking ensemble

212~213

이 방법을 이용해 여러가지 blender를 만들 수 있다. 예를들어 그룹을 둘로 나누지 말고 셋으로 나누어서,
그중 첫번째 그룹을 predictor만드는데 사용
방금 만든 predictor 들의 prediction값을 두번째 그룹의 input으로 사용하여 또 다른 training set을 만듦.
방금 만든 또 다른 training set의 prediction값을 세 번째 그룹의 input으로 사용하여 또 또 다른 training set을 만듦.

Scikit-learn은 위 stacking을 지원하지 않음.