**Chapter 7. Ensemble Learning and Random Forests**

1. **Voting Classifiers**
2. **Bagging and Pasting  
   2.1) Bagging and Pasting in Scikit-Learn  
   2.2) Out-of-Bag Evaluation**
3. **Random Patches and Random Subspaces**
4. **Random Forests  
   4.1) Extra-Trees  
   4.2) Feature Importance**
5. **Boosting  
   5.1) AdaBoost  
   5.2) Gradient Boosting**
6. **Stacking**

**Ensemble Learning and Random Forests**

수천 명의 무작위 사람들에게 복잡한 질문을 한 다음 대답을 모은다고 가정 해보십시오. 대부분의 경우이 집계 된 대답은 전문가의 대답보다 낫습니다. 이것을 (Wisdom of the crowd)라고 합니다. 마찬가지로, 분류기나 회귀 분석기와 같은 예측기의 예측을 집계하면 최상의 개별 예측기를 사용하는 것보다 더 나은 예측을 얻을 수 있습니다.

예측기의 그룹을 앙상블(Ensemble)이라고 하며 이 기술을 Ensemble Learning 이라고 하고, Ensemble Learning Algorithm을 Ensemble Method이라고 합니다.

예를 들어 트레이닝 세트의 서로 다른 임의의 서브 세트에 있는 각각의 의사 결정 트리 분류기를 트레이닝 할 수 있습니다. 예측을 하기위해서는 모든 개별 트리에 대한 예측을 얻은 다음 가장 많은 표를 얻는 클래스를 예측한다.

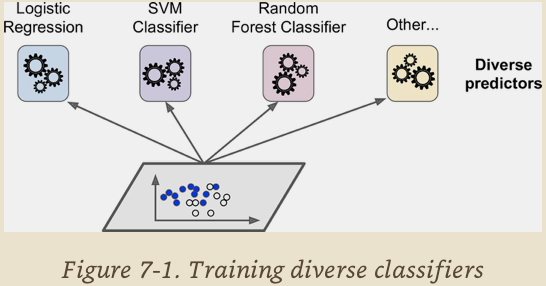
그런 의사 결정 트리의 앙상블은 Random Forest로 불리며, 단순함에도 불구하고 오늘날 가장 유용한 기계 학습 알고리즘 중 하나입니다.

프로젝트의 끝 부분에 이미 Ensemble 메서드를 사용하고, 몇 가지 좋은 예측기를 이미 구축하고 더 나은 예측기로 결합 할 수 있습니다. 실제로, Machine Learning 대회에서 우승 한 솔루션에는 여러 Ensemble 방법 (Netflix Prize 대회에서 가장 유명)이 포함됩니다.

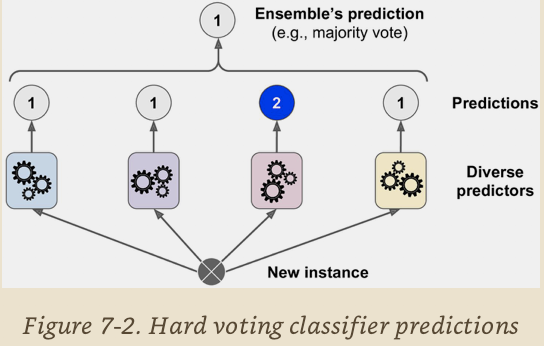
이 장에서는 배깅 (bagging), 부스팅 (boosting), 스태킹 (stacking), 랜덤 포레스트 (Random Forests) 및 기타 몇 가지를 포함하여 가장 많이 사용되는 Ensemble 메서드에 대해 설명합니다.

1. **Voting Classifiers**

당신이 약 80 %의 정확도를 달성하는 몇 가지 분류기를 훈련했다고 가정합니다. Logistic Regression 분류기, SVM 분류기, Random Forest 분류기, K-Nearest Neighbors 분류기 등이 있을 수 있습니다



보다 우수한 분류 기준을 만드는 가장 간단한 방법은 각 분류 기준의 예측을 집계하고 가장 많은 표를 얻는 클래스를 예측하는 것입니다. 이 다수결 분류기는(majority-vote classifier) 하드 투표 분류기(hard voting classifier)라고 불립니다



다소 놀랍게도 이 투표 분류기는 앙상블에서 가장 우수한 분류기보다 더 높은 정확도를 달성하는 경우가 많습니다.

사실 각 분류기가 약한 학습자(weak learner) (무작위 추측보다 약간 나은 것을 의미) 인 경우에도 앙상블은 충분한 수의 약한 학습자가 있고 충분히 다양하다면 ​​강력한 학습자(strong learner)(높은 정확도 달성)일 수 있습니다.

다음 비유는 이 신비에 대해 밝히는 데 도움이 될 수 있습니다.

헤드가 나올 확률이 51 %이고 꼬리가 올 가능성이 49 % 인 약간 편향된 동전이 있다고 가정합니다. 동전을 던지면서 머리의 비율이 머리의 확률에 더 가까워집니다 (51 %).

그림 7-3은 10 개의 일련의 편향된 동전 던지기를 보여줍니다. 토스 수가 늘어 나면 머리의 비율이 51 %에 도달한다는 것을 알 수 있습니다. 궁극적으로 모든 10번의 던지기는 51 %에 가깝게 끝나 50 %를 지속적으로 초과합니다.

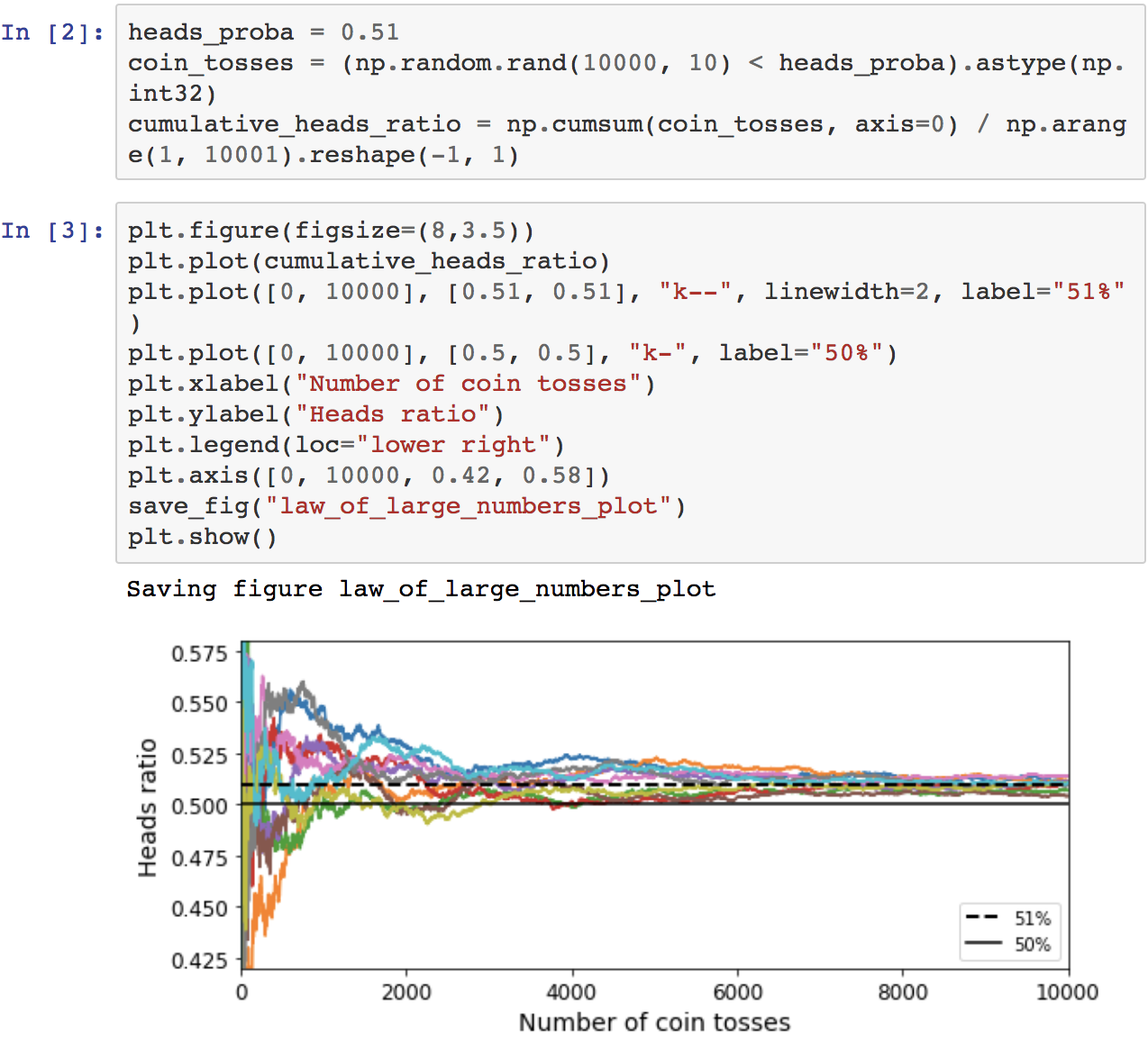
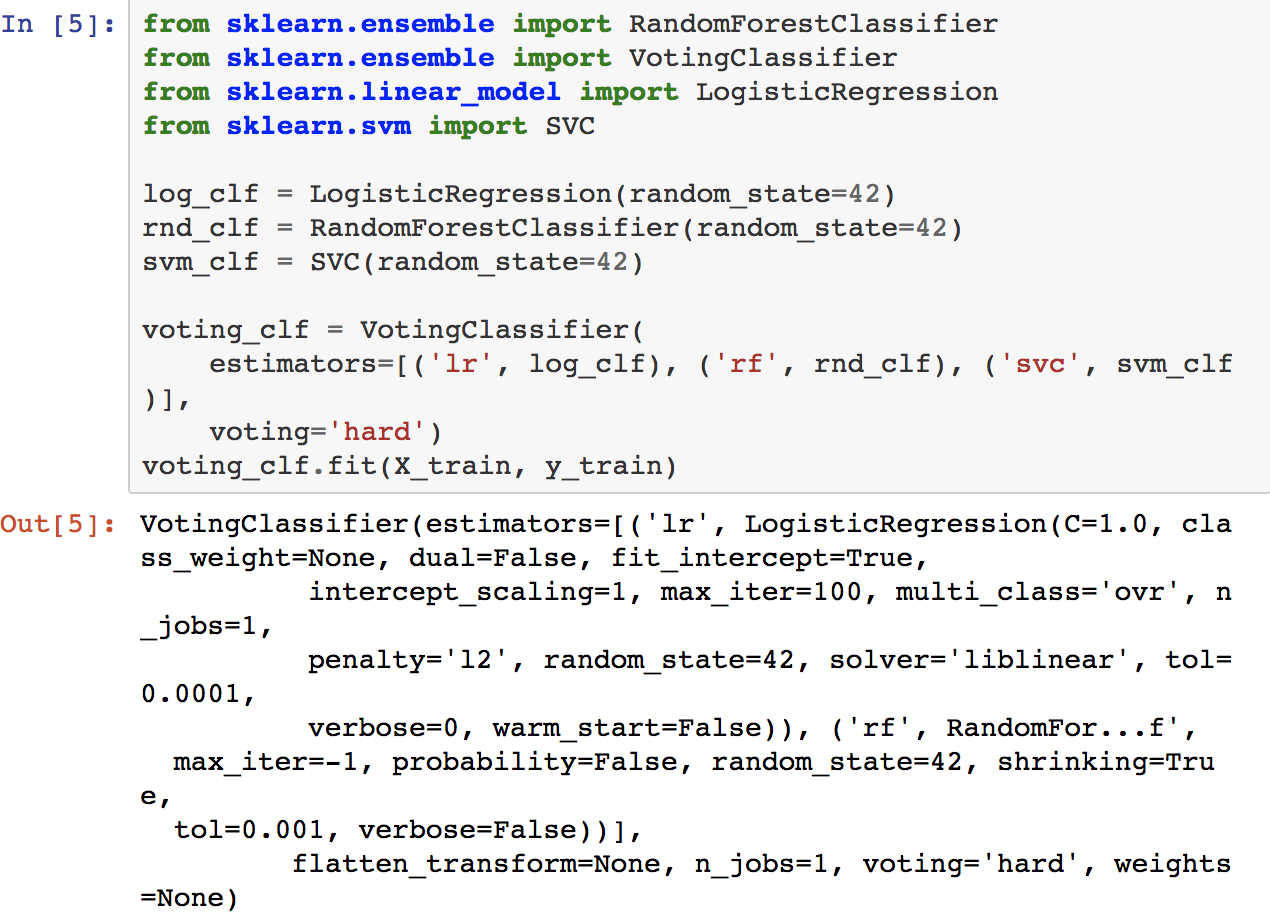


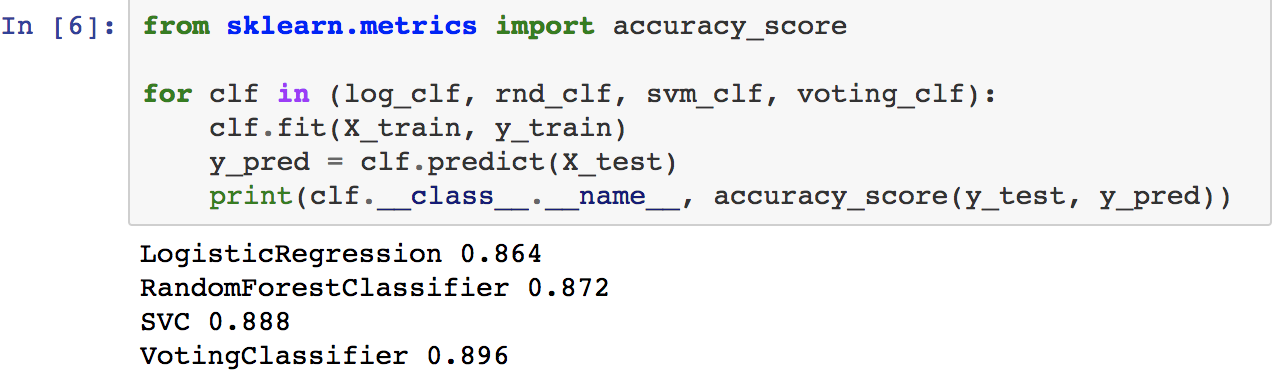
그림 7-3

마찬가지로, 시간의 51 % 만 개별적으로 수정하는 1,000 개의 분류기를 포함하는 앙상블을 작성한다고 가정합니다. 과반수로 투표한 클래스를 예상한다면 최대 75 %의 정확성을 기대할 수 있습니다! 그러나 이것은 모든 분류기들이 완전히 독립적이어서 관련 없는 동일한 오류가 발생하는 경우에만 해당되며 이는 동일한 데이터에 대해 교육을 받았기 때문에 분명하지 않습니다. 그들은 같은 유형의 오류를 만들 가능성이 높으므로 잘못된 수업에 대해 다수 표를 얻어 앙상블의 정확성을 감소시킵니다.

Ensemble 메서드는 예측기가 서로 독립적 일 때 가장 잘 작동합니다. 다양한 분류기를 얻는 한 가지 방법은 매우 다른 알고리즘을 사용하여 분류기를 훈련시키는 것입니다. 이는 앙상블의 정확도를 향상시켜 매우 다른 유형의 오류를 만들 가능성을 높입니다.

다음 코드는 Scikit-Learn에서 3 개의 다양한 분류기로(트레이닝 세트는 5 장에서 소개 달 데이터 세트)로 투표 분류기를 생성하여 트레이닝하고 테스트 세트에서 각 분류기의 정확도를 살펴 보겠습니다



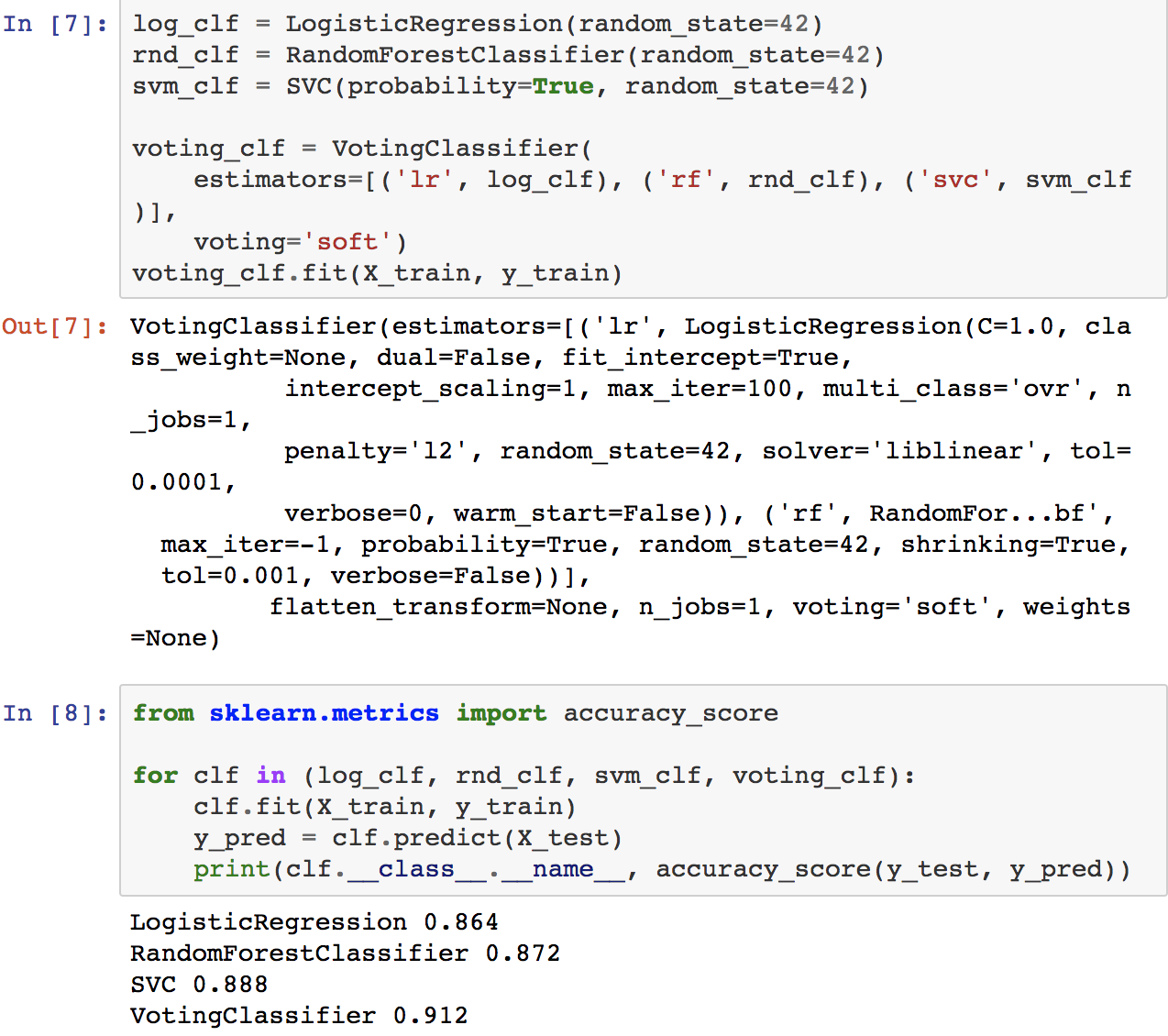


투표 분류기는 모든 개별 분류기보다 약간 우수합니다.

모든 분류기가 클래스 확률을 예측할 수 있는 경우(예 : predict\_proba () 메소드가 있는 경우), Scikit-Learn은 모든 개별 분류 기준에 대해 평균을 낸 클래스 확률이 가장 높은 클래스를 예측하게 지원 합니다. 이를 소프트 투표 (soft voting)라고 합니다.

Soft voting은 매우 확신이 있는 투표에 더 많은 가중치 부여하기 때문에 종종 하드 투표보다 더 높은 성과를 얻습니다. 이것은 기본적으로 SVC 클래스의 경우가 아니므로 확률 하이퍼 매개 변수를 True로 설정해야 합니다.

위의 코드를 수정하여 소프트 투표를 사용하면 투표 분류기가 91 % 이상의 정확도를 달성하게됩니다.

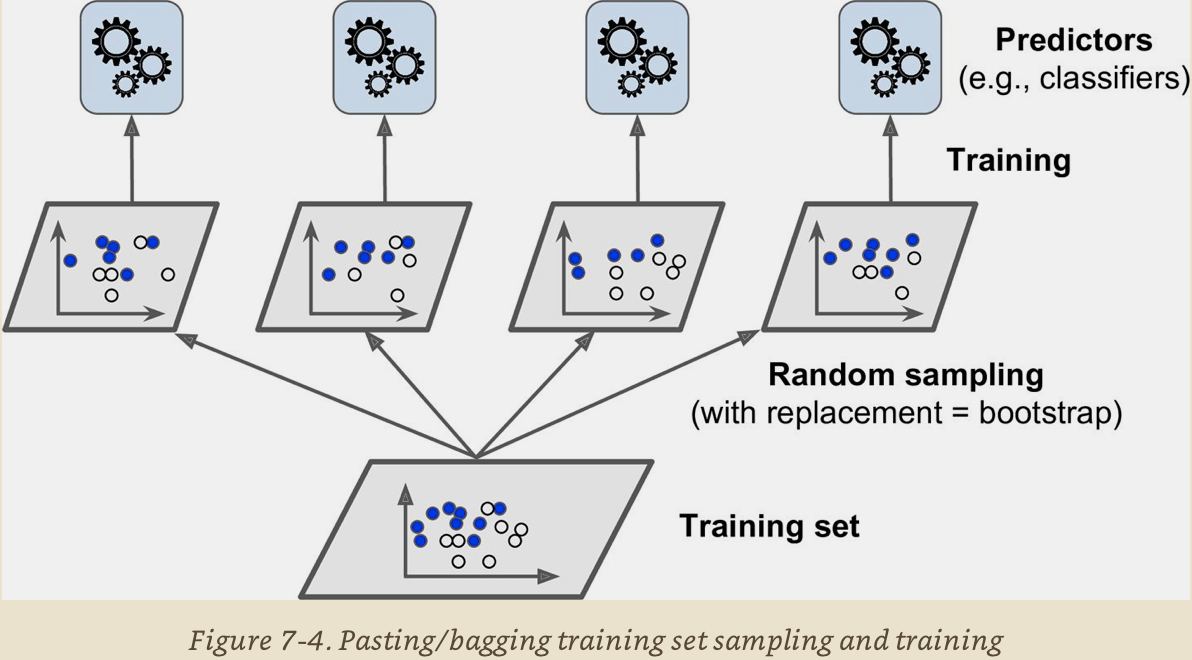


1. **Bagging and Pasting**

다양한 분류 기준을 얻는 한 가지 방법은 방금 설명한 것처럼 매우 다른 트레이닝 알고리즘을 사용하는 것입니다. 다른 접근법은 모든 예측 자에 대해 동일한 학습 알고리즘을 사용하는 것이지만, 트레이닝 세트의 상이한 임의의 부분 집합에 대해 이들을 훈련시키는 것이다.

* replacement가 있을 때 샘플링을 수행하는 경우 이 방법을 bagging (bootstrap aggregating)라고 합니다.
* replacement가 없이 샘플링을 수행 할 때 이를 pasting이라고 합니다.

두 방법 다 샘플링이 여러번 가능하지만 같은 예측기에서 여러 번 중복해서 샘플링을 하면 bagging 이라고 한다.



모든 예측 자의 학습이 완료되면 앙상블은 모든 예측 자의 예측을 단순히 집계하여 새로운 인스턴스에 대한 예측을 할 수 있습니다.

각 개별 예측기는 원래의 트레이닝 세트에서 트레이닝된 것보다 높은 bias을 가지지 만 집계(aggregation)는 bias(편차)와 variance(분산)을 모두 줄입니다. 일반적으로 최종 결과는 앙상블이 원래의 트레이닝 세트에서 훈련된 단일 예측기보다 유사한 bias이지만 더 낮은 variance을 가짐을 의미합니다.

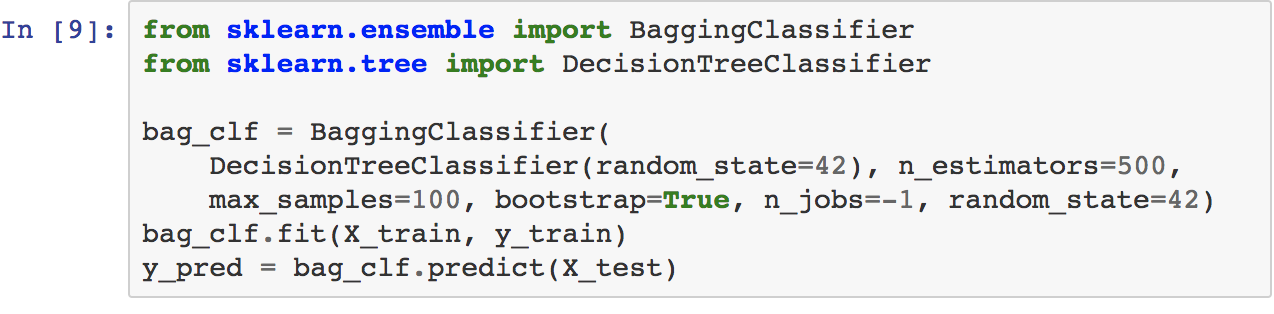
예측기는 서로 다른 CPU 코어 또는 다른 서버를 통해 병렬로 교육 할 수 있습니다. 마찬가지로, 예측을 병렬로 수행 할 수 있습니다. 이것은 bagging 및 pasting이 널리 사용되는 이유 중 하나입니다.

**2.1)** **Bagging and Pasting in Scikit-Learn**

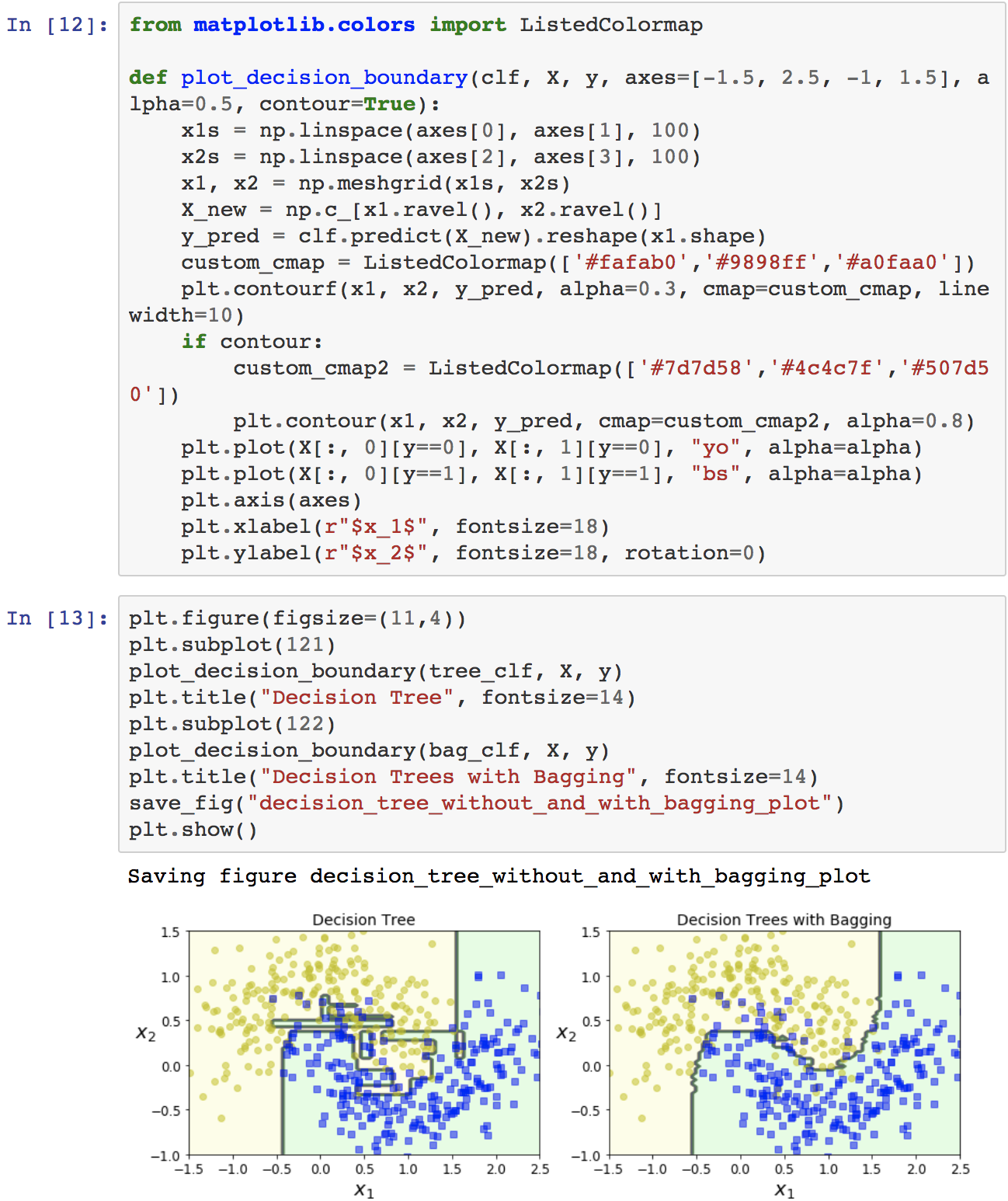
Scikit-Learn는 BaggingClassifier 클래스(또는 회귀 분석을 위한 BaggingRegressor)를 사용하여 bagging 및 pasting모두를 위한 간단한 API를 제공합니다.

다음 코드는 트레이닝 세트에서 무작위로 샘플링 한 100 개의 트레이닝 인스턴스에 대해 트레이닝 한 500 개의 의사 결정 트리 분류기를 500 개의 앙상블로 구성합니다. (이 방법은 bagging의 예이고 pasting을 사용하려면 bootstrap = False로 설정)

. n\_jobs 파라미터는 Scikit-Learn에 학습 및 예측에 사용할 CPU 코어 수를 알려줍니다(-1 사용 가능한 모든 코어 사용을 지시 함).



[BaggingClassifier는 기본 분류기가 의사 결정 트리 분류기의 경우인 클래스 확률을 예측할 수 있는 경우(즉, predict\_proba () 메소드가있는 경우) 하드 투표 대신 소프트 투표를 자동으로 수행합니다.]



위 그림 7-5는 트레이닝된 moons data set에 대한 Decision tree의 Decision boundary와 앞의 코드에서 500개의 트리로 구성된 bagging 앙상블의 Decision boundary를 비교합니다.

보시다시피, 앙상블의 예측은 단일 의사 결정 트리의 예측보다 훨씬 더 일반적일 수 있습니다. 즉, 앙상블은 비슷한 bias을 가지지만 variance은 더 적습니다. (훈련 세트에서 거의 동일한 수의 오류를 만들지만 결정 경계는 덜 불규칙합니다).

Bootstrapping은 각 예측기가 트레이닝한 하위 집합에 조금더 다양성을 나타내기 때문에 bagging는 pasting보다 약간 높은 bias로 나타난다. 하지만 예측기의 상관 관계가 낮아져 앙상블의 variance이 줄어들게 됩니다.

이러한 점은 전반적으로 bagging이 종종 더 나은 모델을 만들며 일반적으로 선호하는 이유다.

그러나 여유 ​​시간과 CPU 사용량이 있는 경우 교차 유효성 검사를 사용하여 bagging and pasting를 모두 평가하고 가장 잘 작동하는 것을 선택하십시오.

**2.2) Out-of-Bag Evaluation**

bagging을 사용하면 일부 인스턴스는 주어진 예측기에 대해 여러 번 샘플링 될 수 있지만 다른 인스턴스는 전혀 샘플링 되지 않을 수 있습니다.

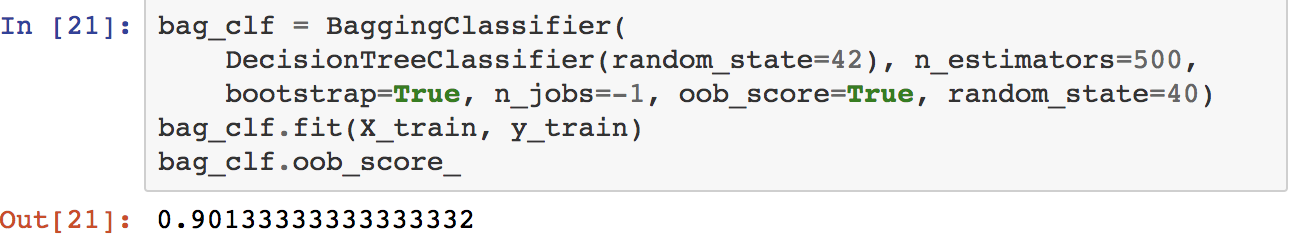
기본적으로 BaggingClassifier는 m 개의 트레이닝 인스턴스를 replacement (bootstrap = True)로 샘플링 합니다. 여기서 m은 트레이닝 세트의 크기입니다.

즉, 교육 인스턴스의 약 63 %만이 각 예측 자에 대해 평균으로 샘플링됩니다. 샘플링되지 않은 교육 인스턴스의 나머지 37 %는 out-of-bag (oob) 인스턴스라고 합니다. 모든 예측 자에 대해 같은 37 %가 아님을 유의합니다.

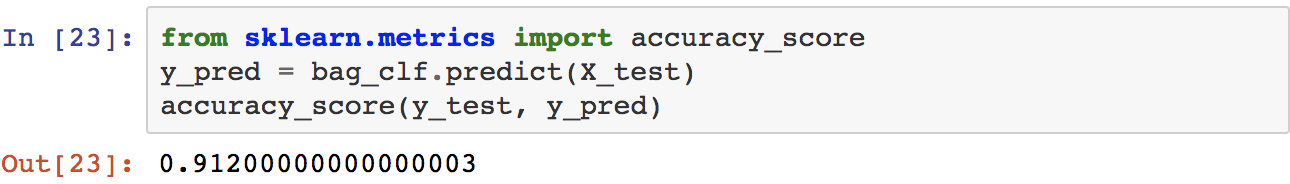
예측기(predictor)는 훈련 중에 oob 인스턴스를 보지 못하기 때문에, 별도의 유효성 검증 세트 또는 cross-validation 필요없이 이러한 인스턴스에서 평가할 수 있습니다.

각 예측기의 oob 평가를 평균하여 앙상블 자체를 평가할 수 있습니다

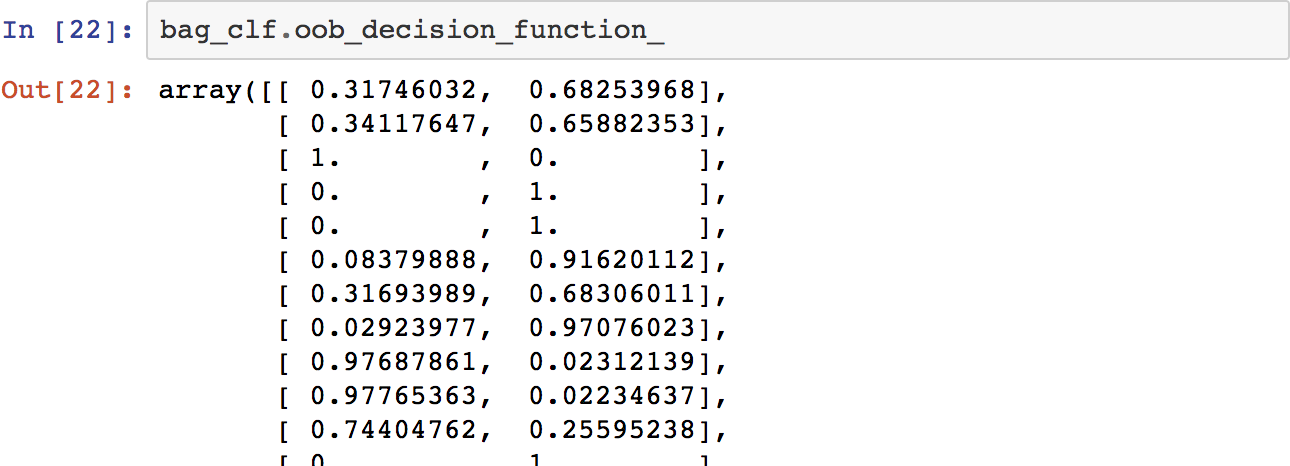
Scikit-Learn에서는 훈련 후 자동으로 oob 평가를 요청하기 위해 BaggingClassifier를 생성 할 때 oob\_score = True를 설정할 수 있습니다.



아래는 별도의 테스트 셋을 통해 나온 결과이다.(비슷한 결과가 나온다.)



각 교육 인스턴스에 대한 oob 결정 기능은 oob\_decision\_function\_ 변수를 통해 사용할 수도 있습니다. 이 경우 (기본 추정기에 predict\_proba () 메소드가 있으므로) 결정 함수는 각 교육 인스턴스에 대한 클래스 확률을 반환합니다.



oob 평가에 따르면 첫 번째 교육 인스턴스는 긍정적 인 클래스에 속할 확률이 68.25 % (그리고 네거티브 클래스에 속한 31.75 %)입니다.

1. **Random Patches and Random Subspaces**

BaggingClassifier 클래스는 기능 샘플링도 지원합니다.

max\_features와 bootstrap\_features의 두 가지 매개 변수로 제어됩니다. 이들은 max\_samples 및 bootstrap과 같은 방식으로 작동하지만 instance 샘플링 대신 feature 샘플링을 수행합니다.

따라서 각 예측자는 입력 특징의 임의의 부분 집합에 대해 학습됩니다.

이것은 특히 고차원 입력 (예 : 이미지)을 다룰 때 유용합니다.

* Random Patches method(임의 패치 방법)  
  트레이닝 인스턴스와 feature을 모두 샘플링하는 것
* Random Subspaces method(랜덤 서브 스페이스 방법)  
  모든 트레이닝 인스턴스(bootstrap = False 및 max\_samples = 1.0)   
  샘플링 feature (즉, bootstrap\_features = True 및 / 또는 max\_features는 1.0보다 작음)

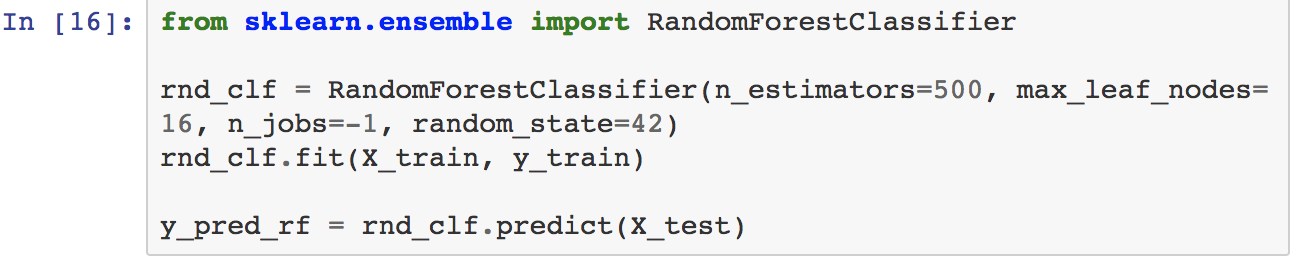
샘플링 기능을 사용하면 더 많은 예측 변수 다양성을 얻을 수 있어 더 낮은 variance에 대한 bias을 더 많이 확보 할 수 있습니다.

1. **Random Forests**

우리가 논의했듯이, Random Forest는 일반적으로 bagging 방법을 통해 트레이닝 된 의사 결정 트리의 앙상블이며, 일반적으로 max\_samples는 트레이닝 세트의 크기로 설정됩니다.

BaggingClassifier를 만들고 DecisionTreeClassifier를 전달하는 대신 RandomForestClassifier 클래스를 사용할 수 있습니다.이 클래스는 Decision Trees에 더 편리하고 최적화되어 있습니다 (마찬가지로 회귀 작업의 경우 RandomForestRegressor 클래스가 있습니다).

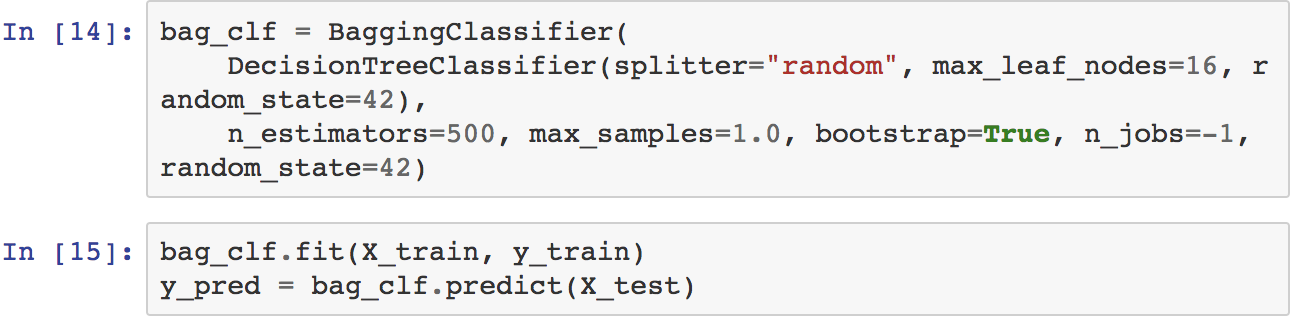
다음 코드는 사용 가능한 모든 CPU 코어를 사용하여 500 개의 트리 (최대 16 개의 노드로 제한된)가있는 RandomForestClassifier를 트레이닝 합니다.



몇 가지 예외를 제외하고, RandomForestClassifier는 DecisionTreeClassifier의 모든 하이퍼 매개 변수 (트리가 어떻게 커지는지 제어)와 BaggingClassifier의 모든 하이퍼 매개 변수를 포함하여 앙상블 자체를 제어합니다.

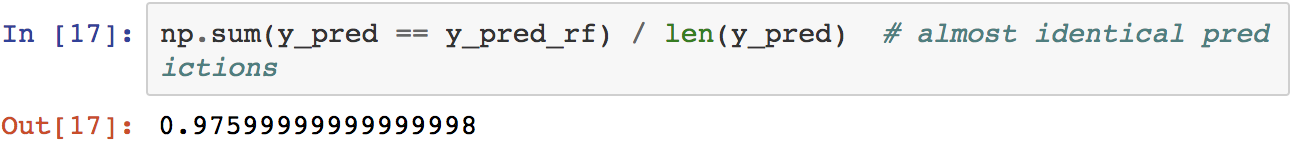
랜덤 포레스트 (Random Forest) 알고리즘은 트리를 성장시킬 때 무작위성을 추가합니다.

노드를 분할 할 때 최상의 feature를 검색하는 대신, feature의 무작위 하위 집합 중에서 가장 좋은 feature를 검색합니다. 이 결과는 더 큰 트리 다양성을 가져 오며, 낮은 variance에 대해 더 높은 bias를 나타내며 전반적으로 더 나은 모델을 생성합니다.



다음 코드에서 BaggingClassifier는 이전의 RandomForestClassifier와 대략 동일합니다.

y\_pred는 BaggingClassifie y\_pred\_rf는 RandomForestClassifier



**4.1) Extra-Trees**

위에서 언급한대로 Random Forest에서 트리를 성장시킬 때 각 노드에서 feature의 임의 하위 집합만 분할용으로 간주하였다.

가장 좋은 임계 값을 검색하는 대신 (일반적인 의사 결정 트리가 하는 것처럼) 각 feature에 임의의 임계 값을 사용하여 트리를 훨씬 더 random으로 만들 수 있습니다.

이러한 트리의 숲은 Extremely Randomized Trees ensemble(Extra-Trees)이라고 불립니다.

* 더 낮은 variance에 대한 더 많은 bias을 나타냅니다.
* Extra-Trees를 일반 Random Forests보다 훨씬 빠르게 트레이닝 할 수 있습니다.  
  (모든 노드에서 각 feature에 대한 최상의 임계 값을 찾는 것이 트리를 성장시키는 데 가장 시간이 많이 걸리는 작업 중 하나이기 때문에 )

Scikit-Learn의 ExtraTreesClassifier 클래스를 사용하여 Extra-Trees 분류기를 만들 수 있습니다. API는 RandomForestClassifier 클래스와 동일합니다. 마찬가지로, ExtraTreesRegressor 클래스는 RandomForestRegressor 클래스와 동일한 API를가집니다.

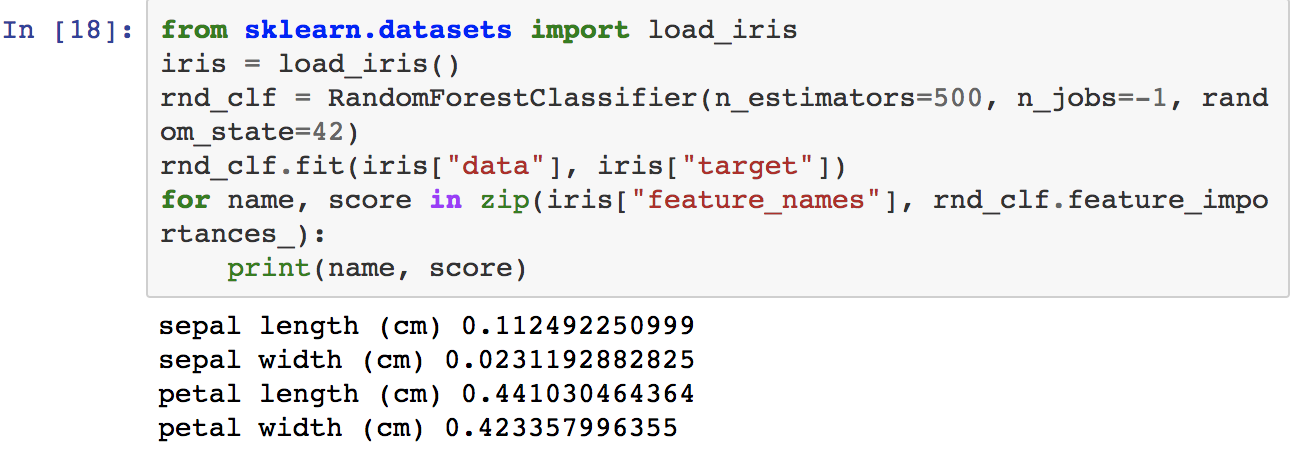
[RandomForestClassifier가 ExtraTreesClassifier보다 좋게 작동하는지 나쁜지를 미리 알려주는 것은 어렵습니다. 일반적으로 알 수 있는 유일한 방법은 두 가지를 시도하고 cross-validation를 사용하여 비교하 (그리드 검색을 사용하여 하이퍼 매개 변수를 조정하는 것입니다).]

**4.2) Feature Importance**

Random Forests의 또 다른 우수한 품질은 각 feature의 상대적 중요성을 쉽게 측정 할 수 있다는 것입니다. Scikit-Learn은 해당 기능을 사용하는 트리 노드가 평균적으로 불순물을 얼마나 줄일 수 있는지 보고하여 feature의 중요성을 측정합니다.

Scikit-Learn은 훈련 후 각 feature에 대해 이 점수를 자동으로 계산 한 다음 모든 feature의 합이 1이되도록 결과를 조정(scale)합니다. feature\_importances\_ 변수를 사용하여 결과에 접근 할 수 있습니다.

예로 다음 코드는 붓꽃(Iris) 데이터 세트의 RandomForestClassifier를 학습하고 각 feature의 중요성을 나타냅니다.

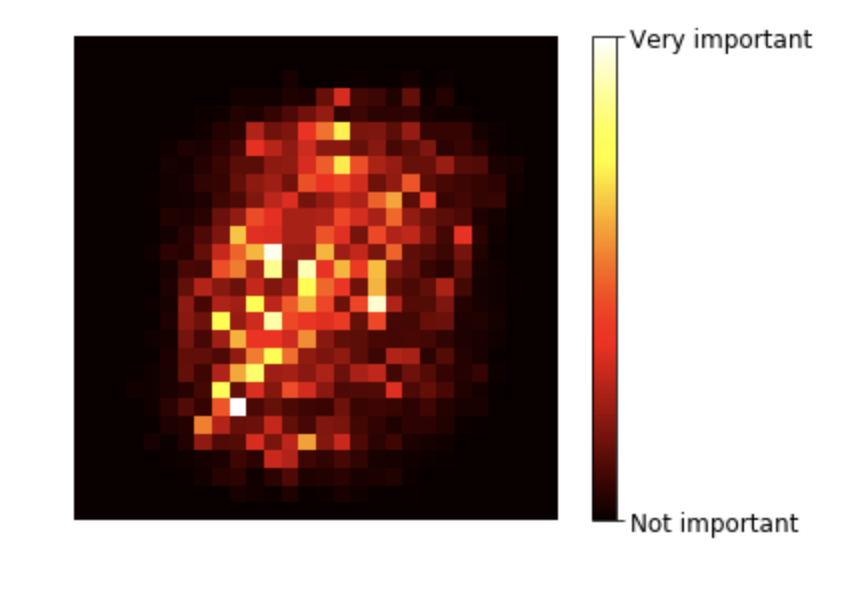


꽃잎 길이 (44 %)와 너비 (42 %)가 가장 중요한 특징 인 반면,

꽃받침 길이(11 %)와 너비(2%)는 비교적 중요하지 않은 것으로 나타났습니다.

마찬가지로, MNIST 데이터 세트에서 RandomForestClassifier를 훈련시키고 각 픽셀의 중요성을 플롯으로 나타내면 그림 7-6에 표시된 이미지를 얻게 됩니다.





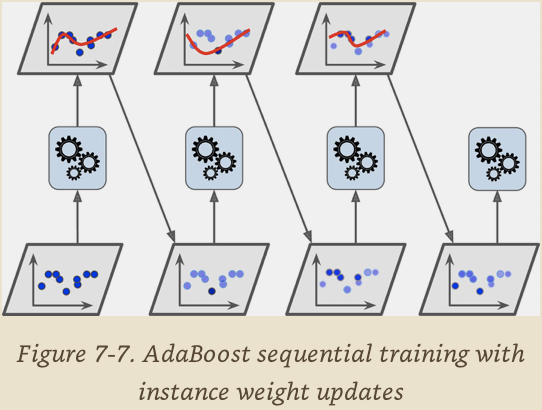
1. **Boosting**

Boosting (원래 가설 증폭이라고 부름)은 여러 가지 weak learners를 strong learner로 결합 할 수 있는 Ensemble 방법을 나타냅니다. 대부분의 방법을 향상시키는 일반적인 아이디어는 각자 이전의 것을 수정하면서 순차적(sequentially)으로 예측기를 학습하는 것입니다.

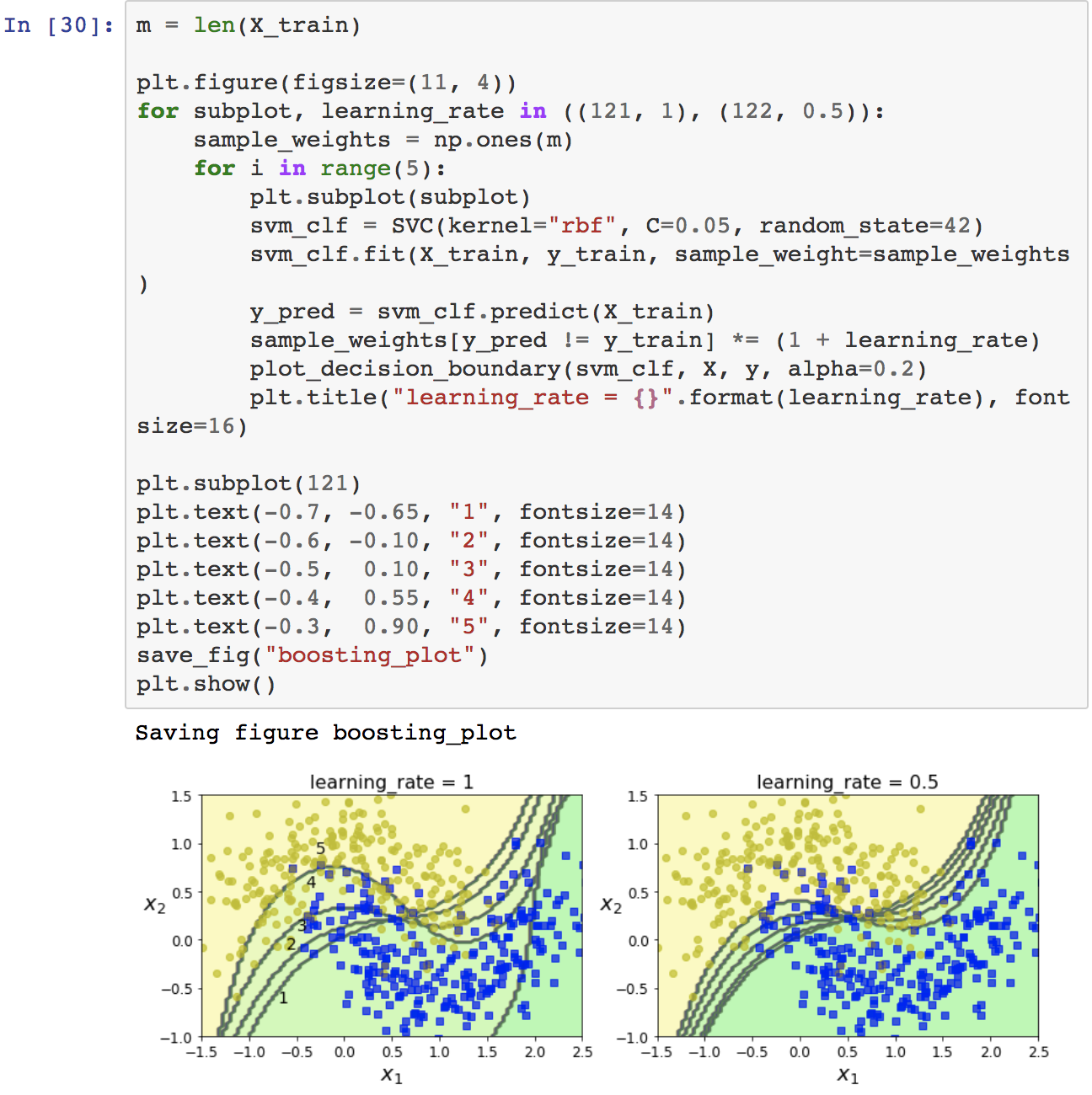
사용 할 수 있는 것에 여러 가지가 있지만 가장 널리 사용되는 것은 AdaBoost(Adaptive Boosting)과 Gradient Boosting입니다.

**5.1) AdaBoost**

새로운 예측기의 predecessor(전에 했던 것)를 수정하는 한 가지 방법은 전임자가 수행한 트레이닝 인스턴스에 조금 더 주의를 기울이는 것입니다. 그 결과 하드 케이스에 점점 더 초점을 맞추는 새로운 예측기가 생깁니다.(점점 더 견고해진다)이것은 AdaBoost에서 사용하는 기법입니다.



예를 들어, AdaBoost 분류기를 작성하려면 의사 결정 트리와 같은 첫 번째 기본 분류기가 학습되어 훈련 세트에 대한 예측을 수행하는 데 사용됩니다. 잘못 분류된 트레이닝 인스턴스의 상대적 가중치가 증가합니다. 두 번째 분류기는 업데이트 된 가중치를 사용하여 트레이닝되며 다시 트레이닝 세트에 대한 예측을 수행하고 가중치가 업데이트됩니다



위 그림 7-8은 moons 데이터에서 5 개의 연속 예측 자의 결정 경계를 보여줍니다.

첫 번째 분류기는 잘못된 인스턴스를 많이 가져 오므로 가중치가 높아집니다. 따라서 두 번째 분류기는 이러한 인스턴스에 대해 더 나은 작업을 수행합니다.

오른쪽의 플롯은 학습 속도가 절반으로 되는 것을 제외하고는 동일한 예측기 순서를 나타낸다 (즉, 오분류된 인스턴스 가중치는 모든 반복에서 절반만큼 증가한다).

이 순차학습 기술은 비용 함수를 최소화하기 위해 단일 예측 변수를 조정하는 대신 AdaBoost가 앙상블에 예측 변수를 추가하고 점진적으로 개선하는 점을 제외하면 Gradient Descent와 일부 유사합니다.

모든 예측 변수가 훈련되면 앙상블은 가중치 부여된 학습 세트의 전체 정확도에 따라 예측 변수가 다른 가중치를 제외하고는 bagging 또는 pasting와 매우 유사한 예측을 합니다.

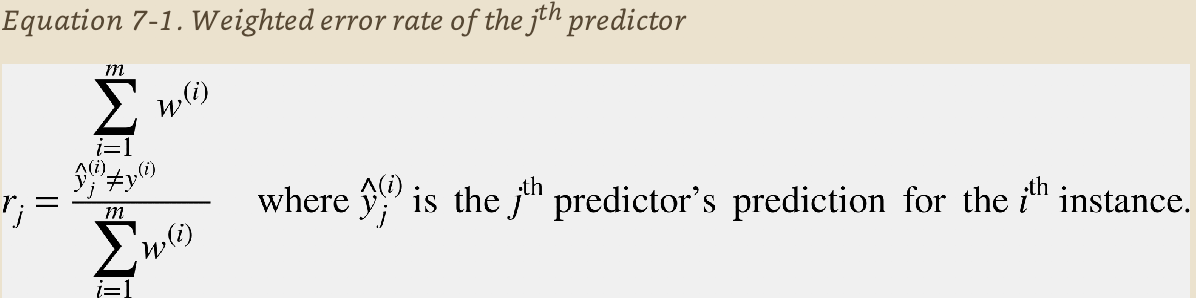
이 순차 학습 기술에는 한 가지 중요한 단점이 있습니다. 각 예측기가 이전 예측기를 학습하고 평가한 후에 만 ​​학습 할 수 있기 때문에 병렬화 할 수 없습니다.

결과적으로, bagging이나 pasting뿐만 아니라 확장되지 않습니다.

AdaBoost 알고리즘을 자세히 살펴 보겠습니다. (식 7-1)

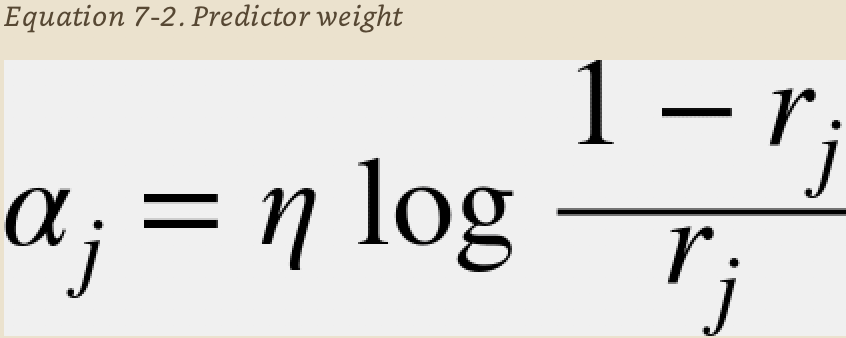
각 인스턴스 가중치 w(i)는 처음에 1/m으로 설정 됩니다.

제 1 예측기는 트레이닝되고, 그 가중 된 에러율 r1은 트레이닝 세트에서 계산된다



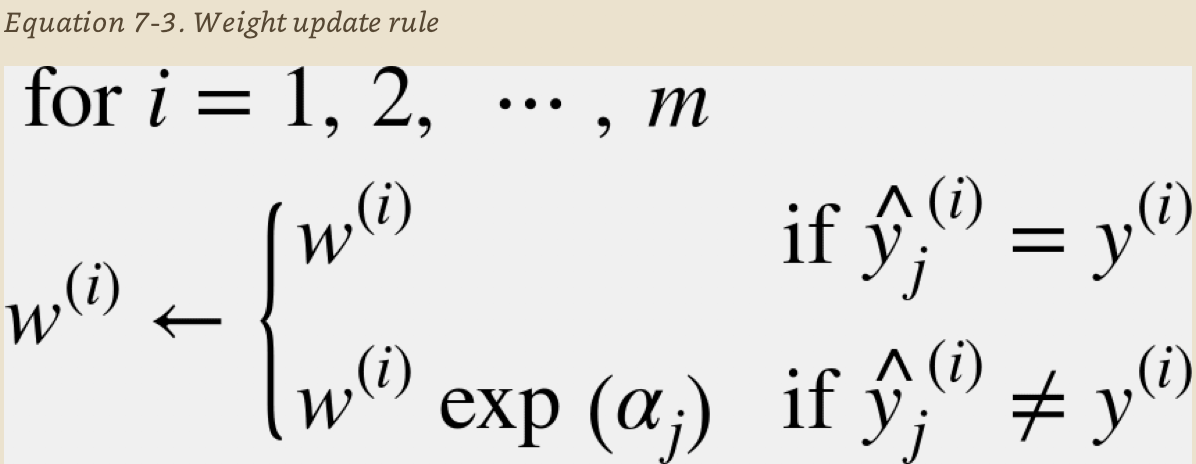
식 (7-2)를 사용하여 예측 변수의 가중치 αj를 계산한다.

여기서 η은 learning-rate 하이퍼파라미터 (기본값은 1)이다. 예측기가 정확해질수록 그 가중치는 더 커집니다. 무작위로 추측하는 경우 가중치가 0에 가까워집니다. 그러나 임의 추측보다 정확하지 않은 경우 그 무게는 음수가 됩니다.



다음으로 인스턴스 가중치는 식 7-3을 사용하여 업데이트됩니다.

잘못 분류 된 인스턴스가 boost됩니다.

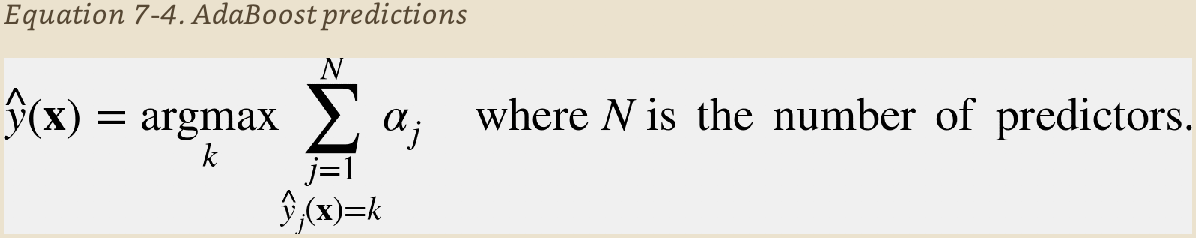


그런 다음 모든 인스턴스 가중치가 정규화 됩니다 (즉, 으로 나눕니다).

마지막으로 업데이트된 가중치를 사용하여 새 예측기를 학습하고 전체 프로세스가 반복됩니다. 알고리즘은 원하는 수의 예측기에 도달하거나 완벽한 예측기가 발견되면 중지합니다.

예측을 수행하기 위해 AdaBoost는 모든 예측기의 예측을 계산하고 예측기 가중치 αj를 사용하여 이들의 가중치를 계산합니다.

예측 된 클래스는 가중 투표의 대다수를 받는 클래스입니다 (식 7-4 참조).



Scikit-Learn은 실제로 SAMME 라는 AdaBoost의 다중 클래스 버전을 사용합니다.

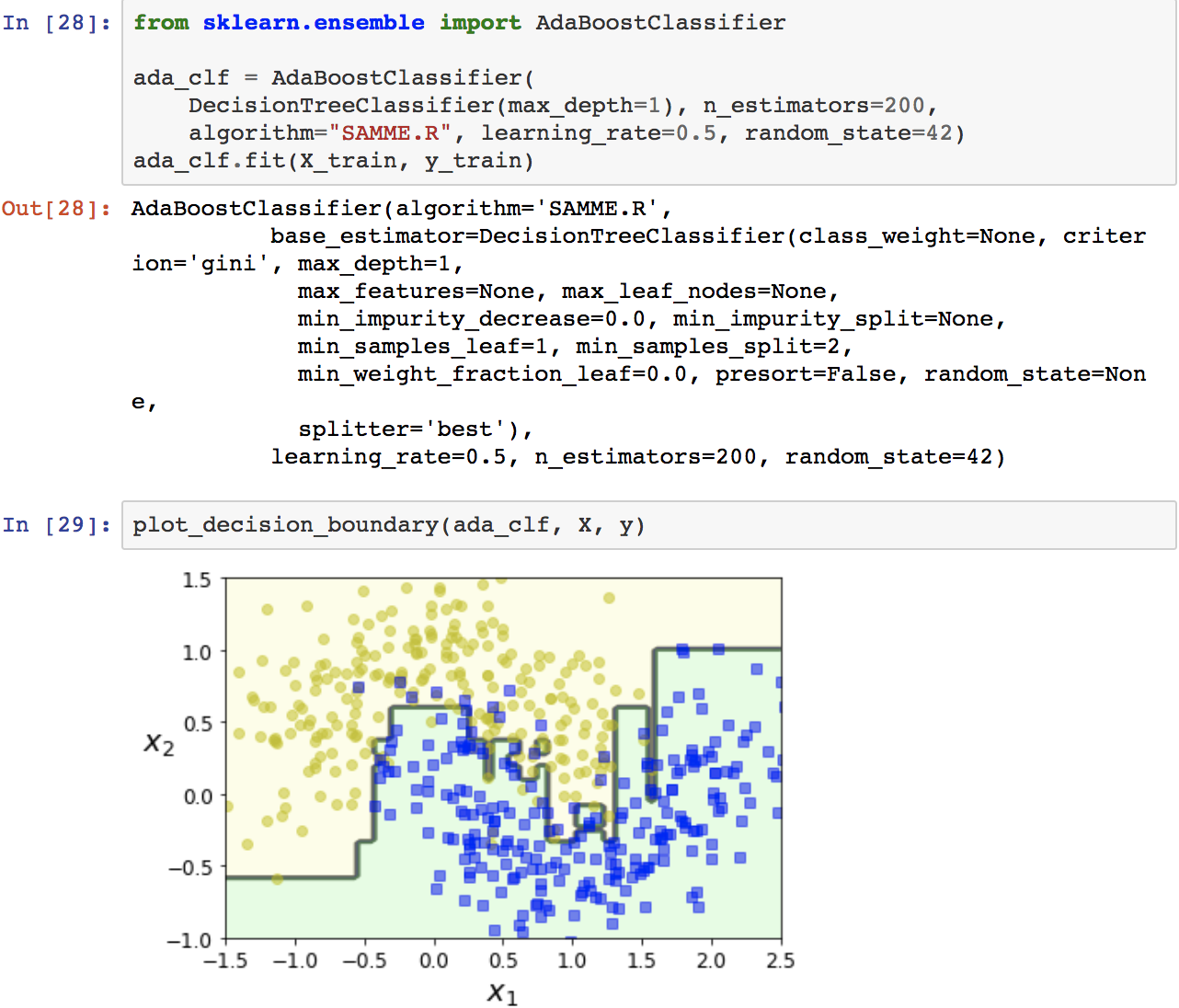
(stands for Stagewise Additive Modeling using a Multiclass Exponential loss function)

클래스가 두 개인 경우 SAMME는 AdaBoost와 동일합니다. 더욱이, 예측기가 클래스 확률을 예측할 수 있다면 (예 : predict\_proba () 메소드가 있는 경우), Scikit-Learn은 SAMME.R (R은 "Real"을 나타냄)이라는 SAMME 변형을 사용할 수 있습니다.이 변형은 클래스 확률 예측보다는 오히려 더 잘 수행됩니다

다음 코드는 Scikit-Learn의 AdaBoostClassifier 클래스를 사용하여 200 개의 Decision Stumps를 기반으로 AdaBoost 분류기를 트레이닝 시킵니다 (예상대로 AdaBoostRegressor 클래스도 있음). Decision Stump는 max\_depth = 1 인 의사 결정 트리입니다.

즉, 단일 결정 노드와 두 개의 잎 노드로 구성된 트리입니다.

다음은 AdaBoostClassifier 클래스의 기본으로 제공되는 추정기(default base estimator)입니다.



AdaBoost 앙상블이 교육 세트에 적합하지 않은 경우 추정기 수를 줄이거나 기본 추정기를 보다 강력하게 정규화 할 수 있습니다.

**5.2) Gradient Boosting**

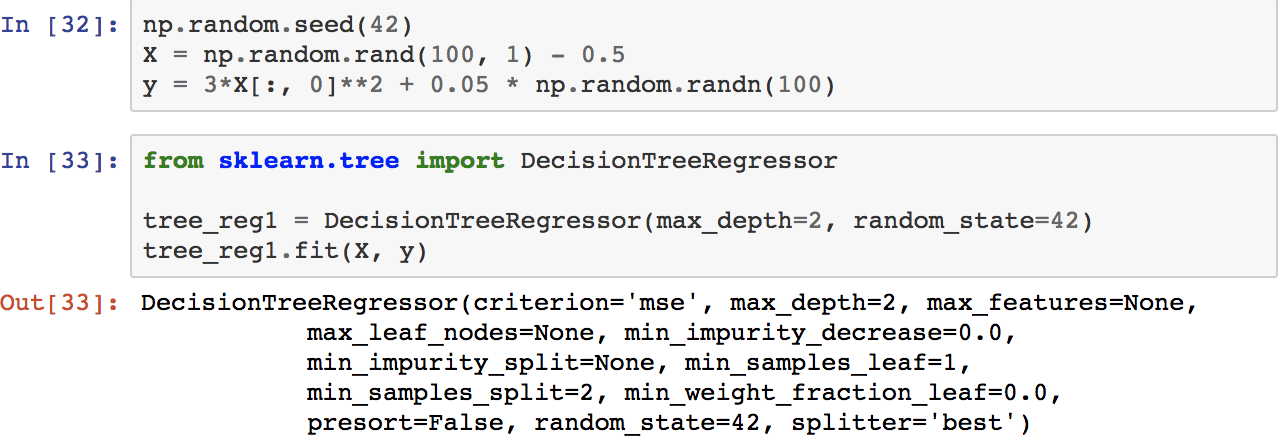
또 다른 인기있는 Boosting 알고리즘은 Gradient Boosting입니다.

AdaBoost와 마찬가지로 앙상블에 예측기를 순차적으로 추가하여 작동합니다. 각자는 이전의 것을 수정합니다. 그러나 AdaBoost와 같이 모든 반복에서 인스턴스 가중치를 조정하는 이전 예측기에 의한 잔여 오류(residual errors)에 새로운 예측기를 맞추려고 합니다.

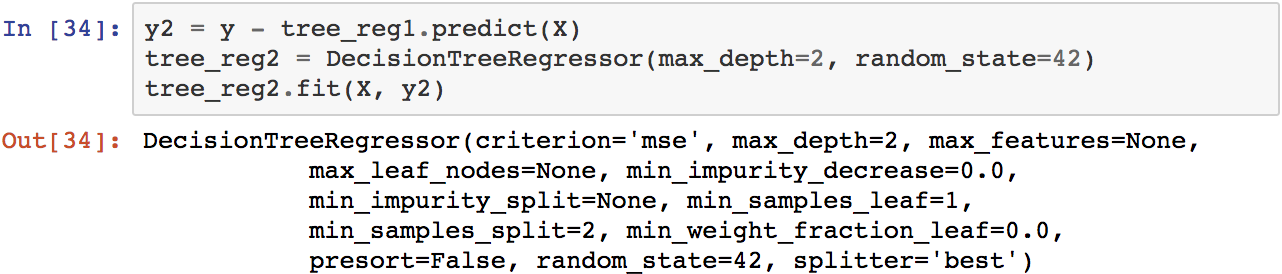
결정 트리를 기본 예측기로 사용하는 간단한 회귀 예제를 살펴 보겠습니다.

이를 Gradient Tree Boosting 또는 Gradient Boosted Regression Trees (GBRT)라고합니다.

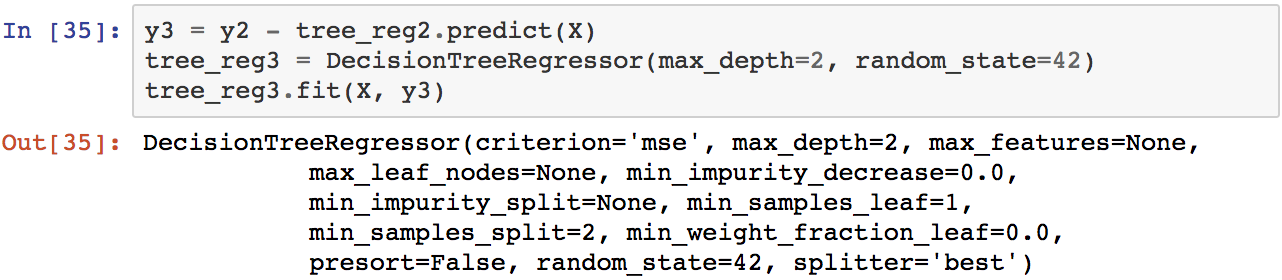
먼저, DecisionTreeRegressor를 트레이닝 세트에 맞춰 보겠습니다.



이제 첫 번째 예측기에 의한 잔여 오류에 대해 두 번째 DecisionTreeRegressor를 훈련 시킨다.



그 다음에 두 번째 예측기에 의한 잔여 오류에 대한 세 번째 회귀 변수를 훈련시킵니다.



이제 세 개의 나무가 포함 된 앙상블이 있습니다. 모든 나무에 대한 예측을 추가하여 새로운 인스턴스에 대한 예측을 할 수 있습니다.

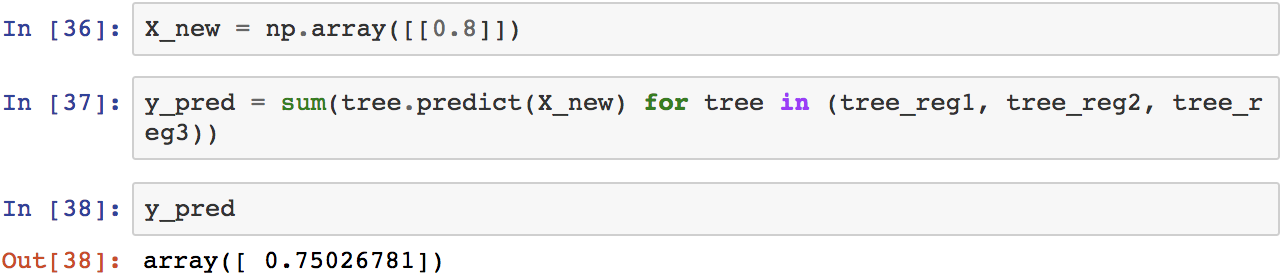


그림 7-9는 왼쪽열에는 세 트리의 예측과 오른쪽 열에는 앙상블의 예측을 나타냅니다.

* 첫 번째 행에서 앙상블은 트리가 하나뿐이므로 예측은 첫 번째 트리의 예측과 같다.
* 두 번째 행에서 첫 번째 트리의 잔여 오류에 대해 새 트리를 학습합니다. 오른쪽에서 앙상블의 예측은 처음 두 트리의 예측 합계와 같습니다.
* 세 번째 행에서 두 번째 트리의 잔여 오류에 대해 다른 트리가 학습됩니다. 트리가 앙상블에 추가됨에 따라 앙상블의 예측이 점점 좋아지는 것을 볼 수 있습니다.

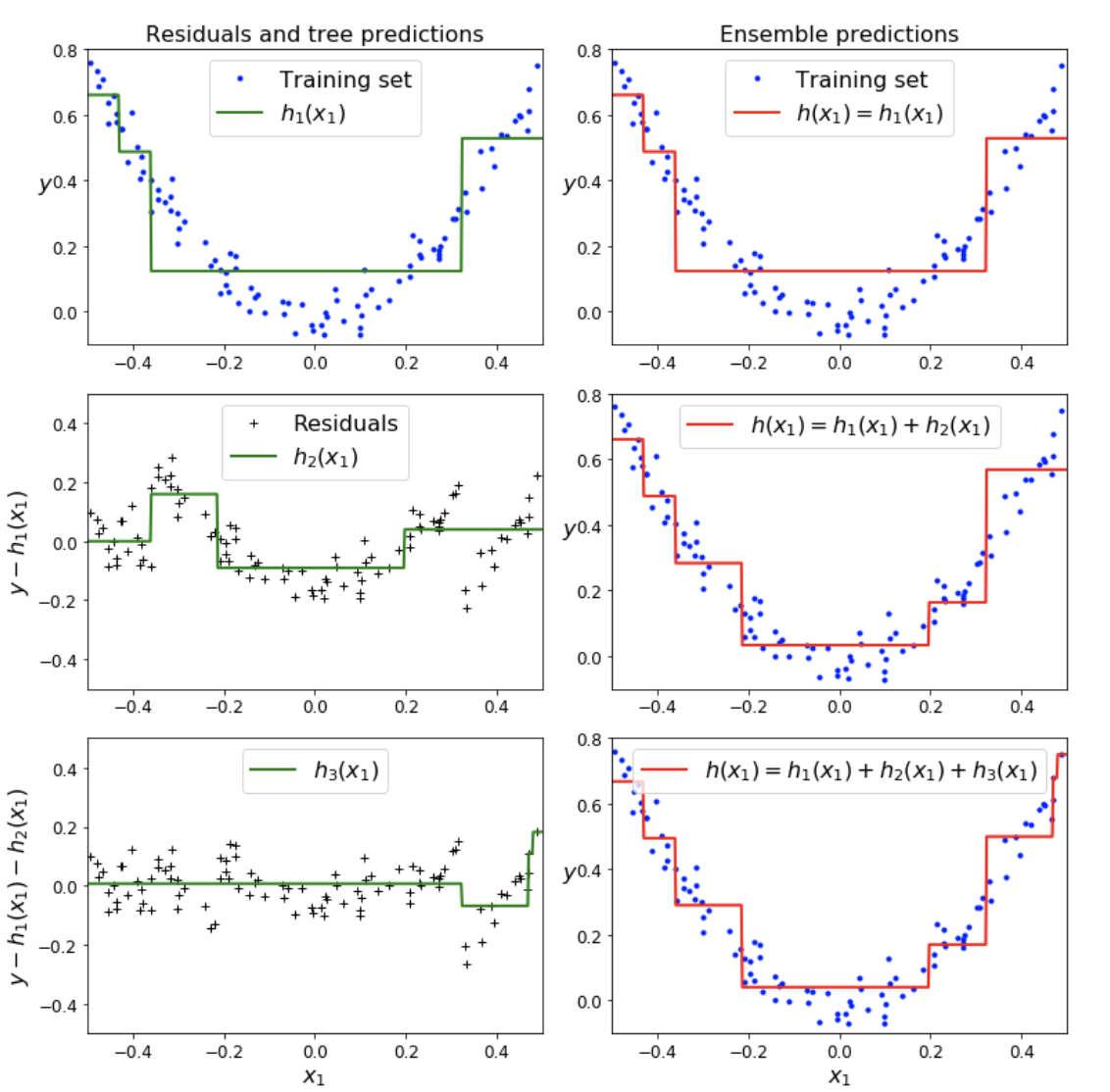
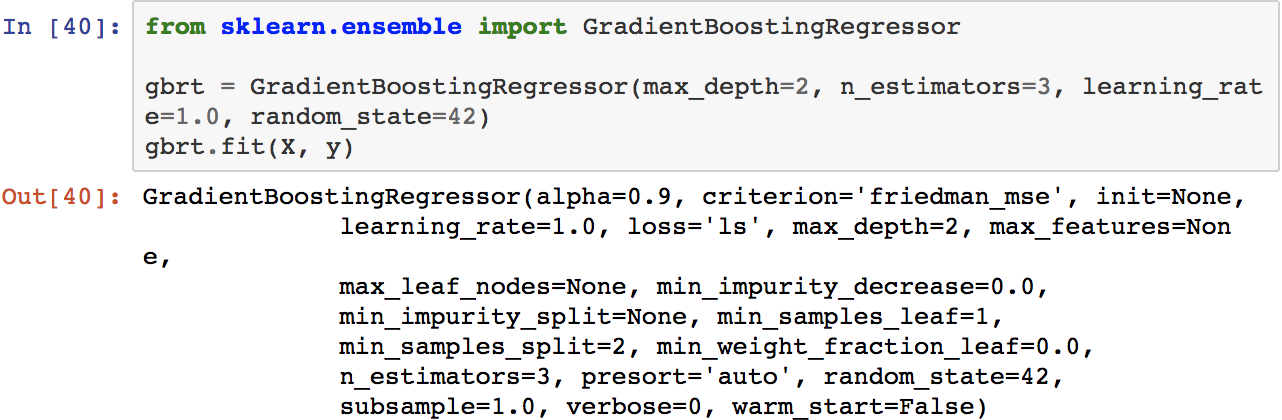


그림 7-9

GBRT 앙상블을 훈련시키는 간단한 방법은 Scikit-Learn의 GradientBoostingRegressor 클래스를 사용하는 것입니다. RandomForestRegressor 클래스와 매우 유사하게, 트리 (n\_estimators)의 수와 같은 앙상블 트레이닝을 제어하는 ​​하이퍼 매개 변수 뿐만아니라 의사 결정 트리의 성장 (예 : max\_depth, min\_samples\_leaf 등)을 제어하는 ​​하이퍼 매개 변수가 있습니다.

다음 코드는 이전과 같은 앙상블을 만듭니다.



learning\_rate 하이퍼 매개 변수는 각 트리의 기여도를 조정합니다. 0.1과 같이 낮은 값으로 설정하면 교육 세트에 맞게 더 많은 트리가 앙상블에 있어야하지만 일반적으로 예측이 잘됩니다. 이것은 shrinkage라고 하는 정규화 기법입니다.

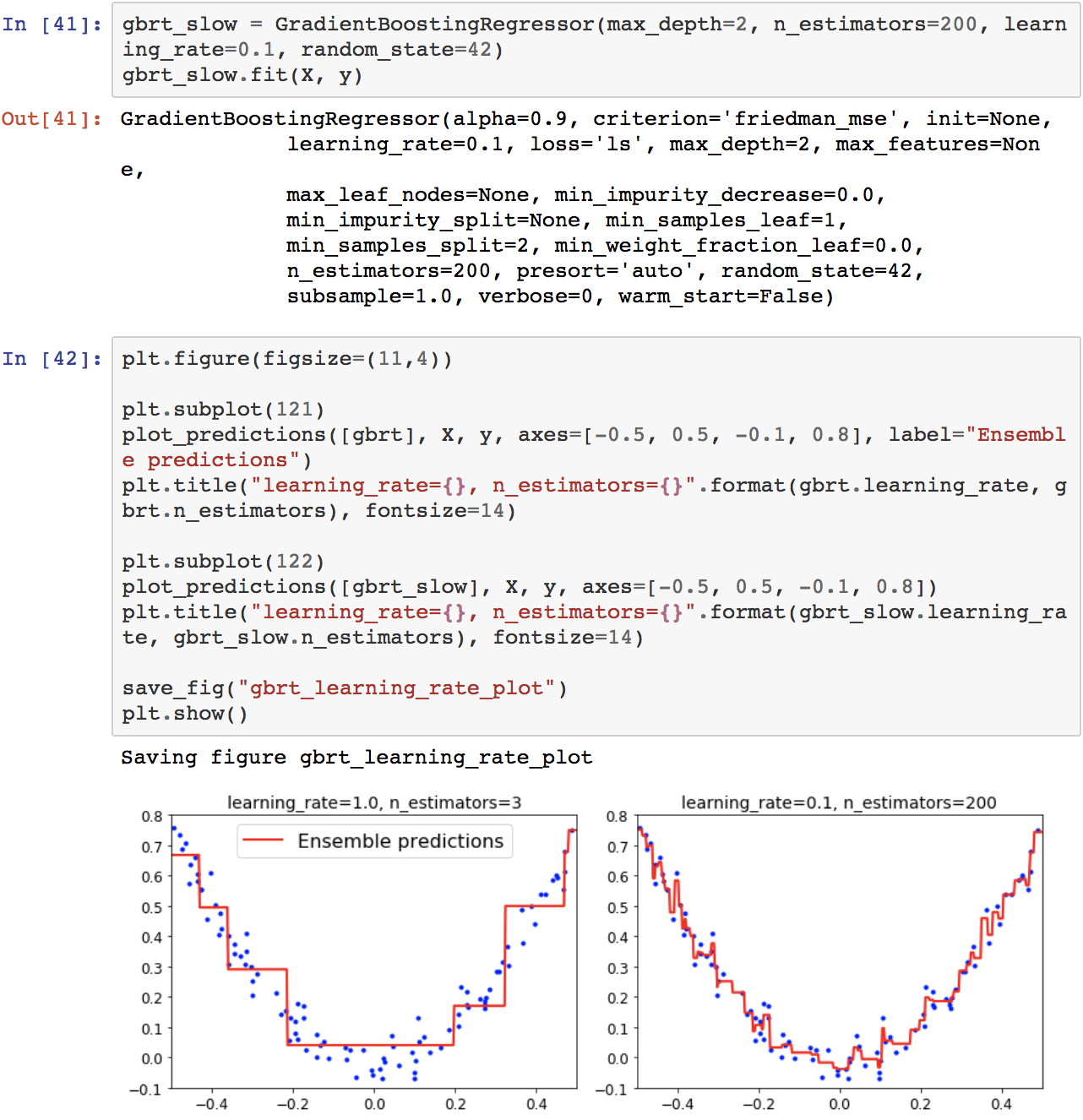


그림 7-10은 ​​낮은 학습률로 훈련된 두 개의 GBRT 앙상블을 보여줍니다.

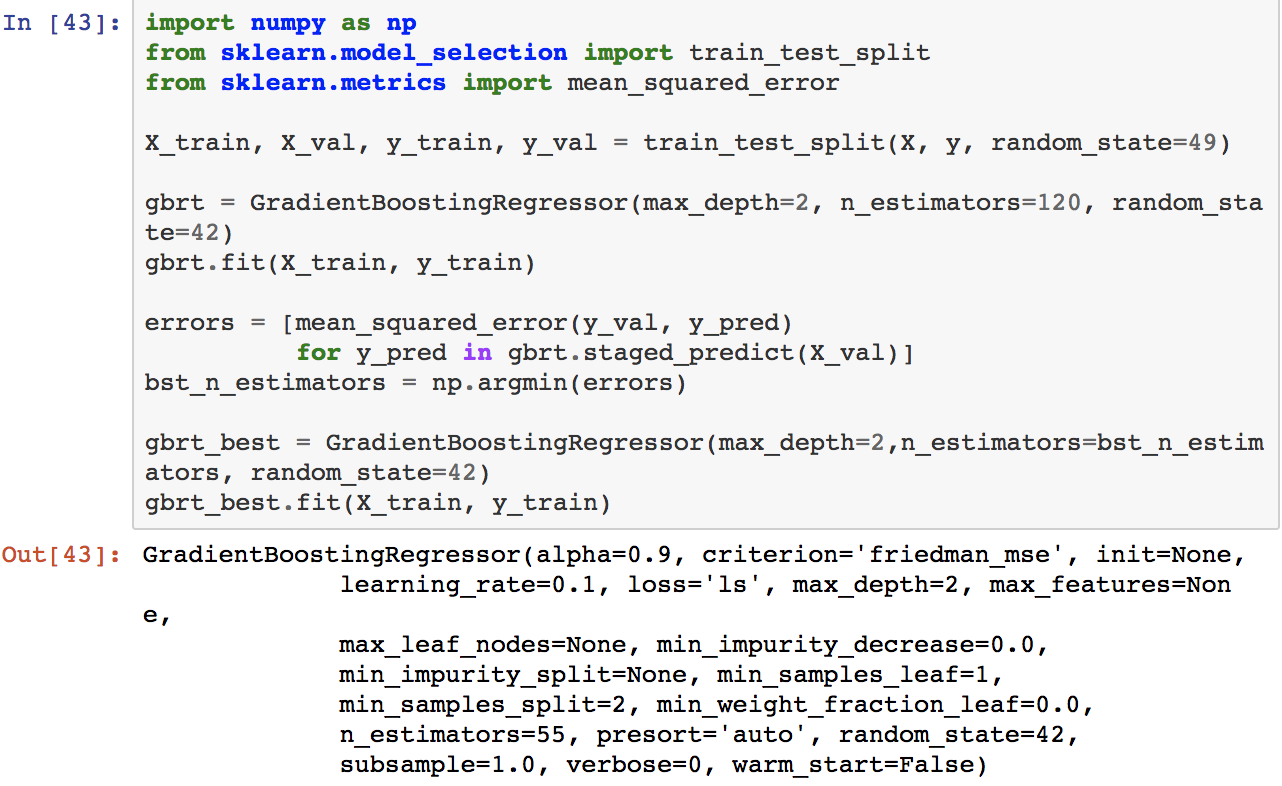
왼쪽에 있는 트리는 트레이닝 세트에 맞는 트리가 충분하지 않은 반면,

오른쪽에 있는 트리는 너무 많고 트레이닝 세트에 적합하지 않습니다.

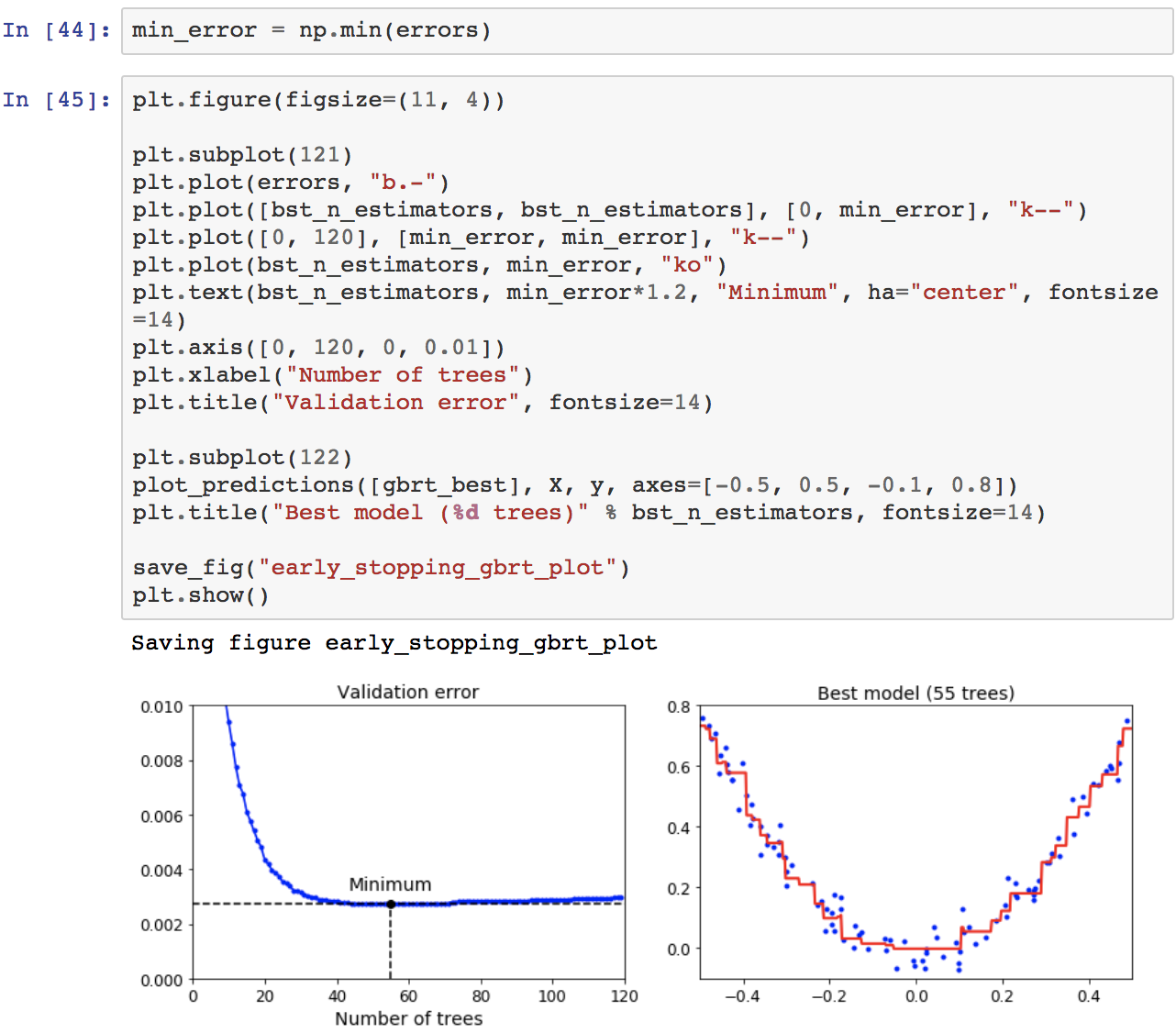
최적의 트리 수를 찾으려면 조기 정지(4 장 참조)를 사용할 수 있습니다.

이를 구현하는 간단한 방법은 staged\_predict () 메소드를 사용하는 것입니다. 하나의 트리, 두 개의 트리 등을 사용하여 각 단계에서 앙상블에 대한 예측을 반복자로 반환합니다.

다음 코드는 120 개의 나무를 가진 GBRT 앙상블을 훈련시킨 다음, 훈련의 각 단계에서 유효성 검사 오류를 측정하여 최적의 트리 수를 찾은 다음 마지막으로 최적의 트리 수를 사용하여 다른 GBRT 앙상블을 훈련시킵니다.

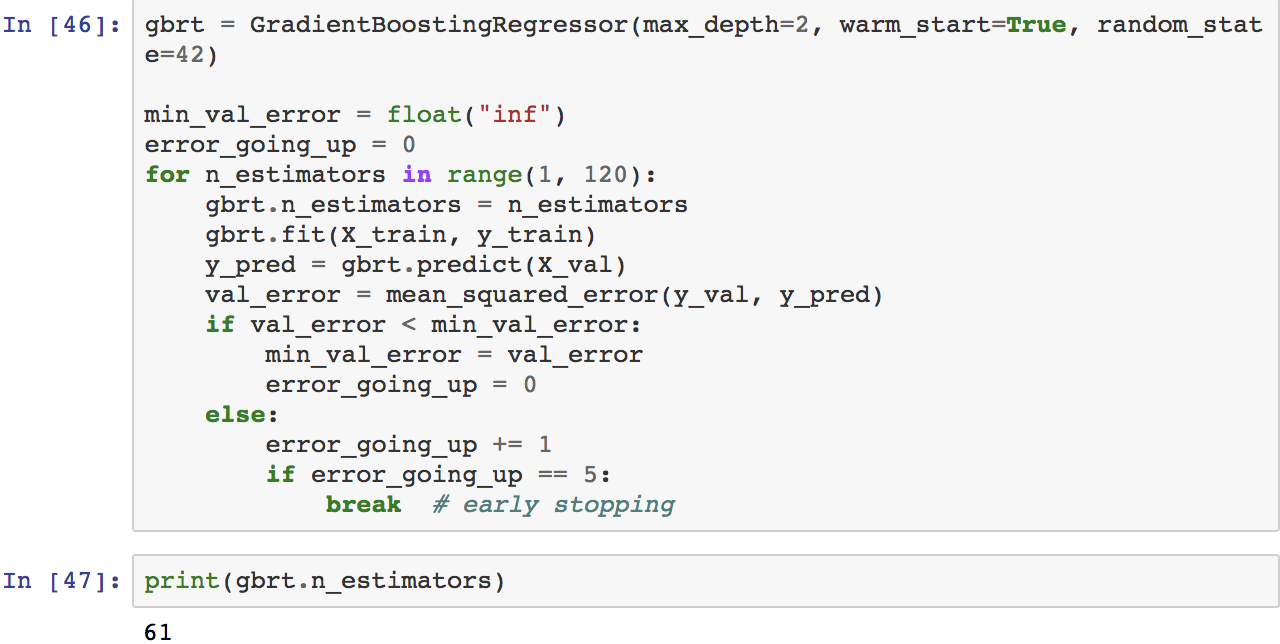


유효성 검사 오류는 그림 7-11의 왼쪽에 표시되며 가장 좋은 모델의 예측은 오른쪽에 표시됩니다.

그림 7-11

또한 일찍 실제로 많은 수의 트리를 훈련시킨 다음 최적의 수를 찾기 위해 일찍 훈련을 중지함으로써 조기 정지를 구현할 수 있습니다. warm\_start = True로 설정하면 fit () 메서드가 호출 될 때 Scikit-Learn에서 기존 트리를 유지하면서 점진적 트레이닝을 허용 할 수 있습니다.

다음 코드는 유효성 검사 오류가 연속 5 회 반복되지 않으면 학습을 중지합니다.



GradientBoostingRegressor 클래스는 각 트리를 트레이닝하는데 사용할 트레이닝 인스턴스의 비율을 지정하는 하위샘플 하이퍼 매개 변수도 지원합니다

예를 들어 subsample = 0.25 인 경우 각 트리는 무작위로 선택된 교육 인스턴스의 25 %에서 학습됩니다. 이것은 더 낮은 bias에 대한 더 높은 variance을 나타냅니다.

그것은 또한 훈련을 상당히 가속화합니다.

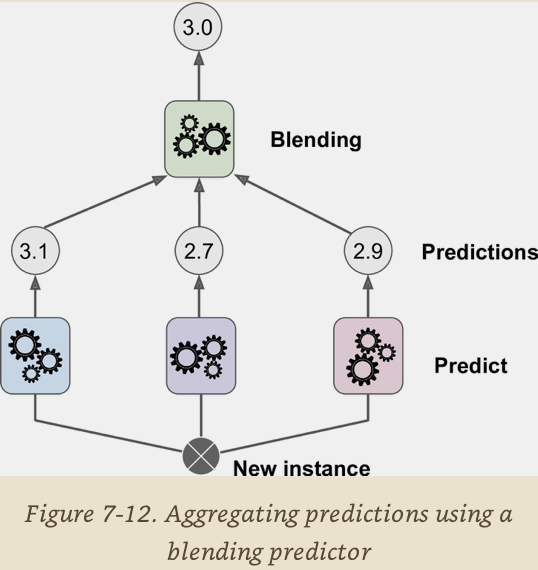
이 기술을 Stochastic Gradient Boosting 이라고 합니다.

1. **Stacking**

이 장에서 논의 할 마지막 앙상블 방법은 stacking (stacked generalization)이라고 합니다.

앙상블에 있는 모든 예측기의 예측을 집계하기 위해 hard-voting같은 사소한 기능을 사용하는 대신,이 집계를 수행하도록 모델을 훈련시키는 것이다.

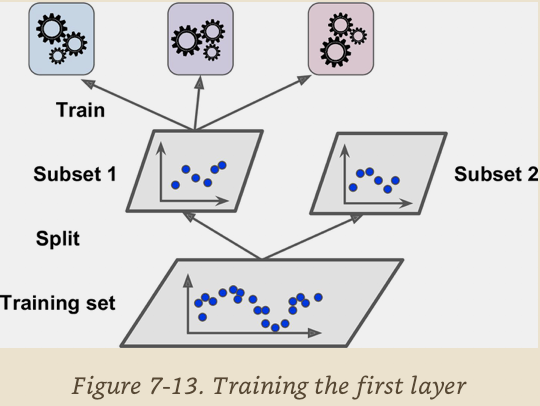
그림 7-12는 새로운 인스턴스에서 회귀 작업을 수행하는 앙상블을 보여줍니다. 하위 3 개의 예측 변수 각각은 다른 값 (3.1, 2.7 및 2.9)을 예측 한 다음 최종 예측기(blender 또는 meta learner)는 이러한 예측을 입력으로 사용하여 최종 예측(3.0)을 만듭니다.



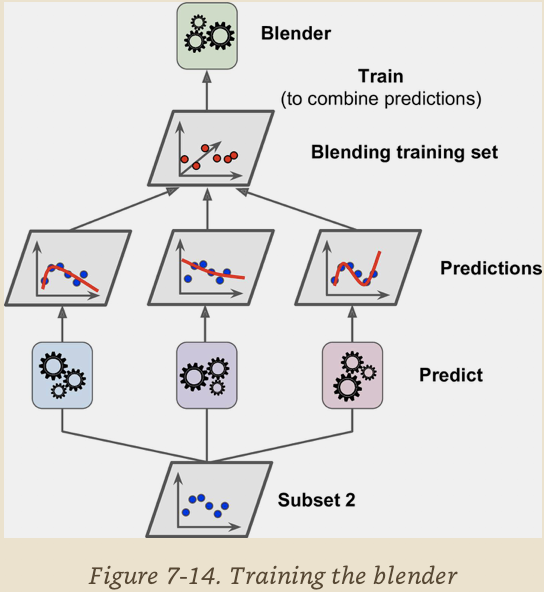
훈련시키기위한 일반적인 방법은 **홀드 아웃(hold-out) 세트**를 사용하는 것입니다.

* 첫째, 트레이닝 세트는 2 개의 부분 집합으로 나뉘어져 있습니다.

첫 번째 계층의 예측기는 첫 번째 하위 세트를 학습하는데 사용됩니다.



* 다음으로, 첫 번째 계층의 예측기는 두 번째(보류 된) 세트에 대한 예측을 학습하는 데 사용됩니다 이렇게 하면 예측기가 트레이닝 중에 이러한 사례를 본 적이 없으므로 예측이 "깨끗합니다".



이제 홀드 아웃 세트의 각 인스턴스에 대해 세 가지 예측 된 값이 있습니다.   
이 예측된 값을 입력 feature로 사용하여 새 트레이닝 세트를 만들면(이 새 교육 세트를 3 차원으로 설정 함) 목표 값(target value)이 유지됩니다.

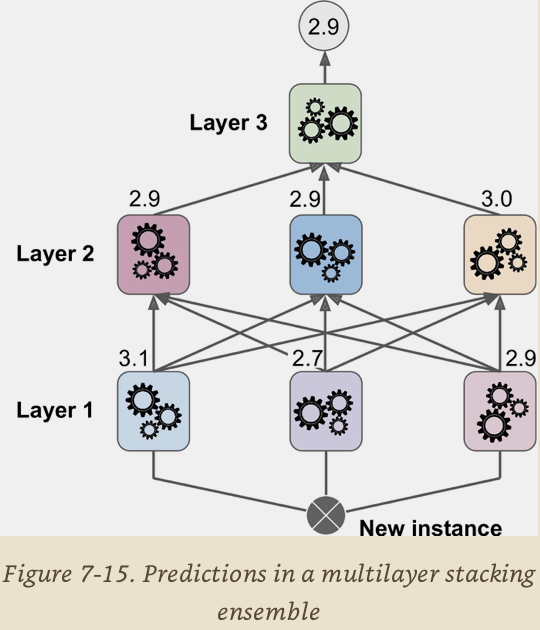
blender는 이 새로운 트레이닝 세트에 대해 학습되고 첫 번째 레이어의 예상치에 따라 목표 값을 예측하는 방법을 학습하게 됩니다.

전체 블렌더 레이어를 얻는 방법으로 여러 가지 블렌더를 실제로 트레이닝 할 수 있습니다 (예 : Linear Regression을 사용하는 블렌더, Random Forest Regression를 사용하는 블렌더 등).

트릭은 트레이닝 세트를 세 개의 하위 세트로 분리하는 것입니다.

* 첫 번째 레이어는 첫 번째 레이어를 학습하는 데 사용되고
* 두 번째 레이어는 두 번째 레이어를 학습하는 데 사용되는 트레이닝 세트를 만드는 데 사용됩니다 (첫 번째 레이어의 예측기에 의해 만들어진 예측을 사용)
* 세 번째 계층은 세 번째 계층을 훈련하기위한 트레이닝 세트를 만드는데 사용됩니다 (두 번째 계층의 예측기에 의해 만들어진 예측을 사용).

이 작업이 완료되면 그림 7-15와 같이 각 계층을 순차적으로 통과하여 새로운 인스턴스에 대한 예측을 할 수 있습니다.



불행하게도 Scikit-Learn은 Stacking을 직접 지원하지 않지만 직접 구현을 수행하는 것은 그리 어렵지 않습니다

다음 연습문제 또는 brew (https : // github.com/ viisar / brew)와 같은 오픈소스를 사용할 수 있습니다.