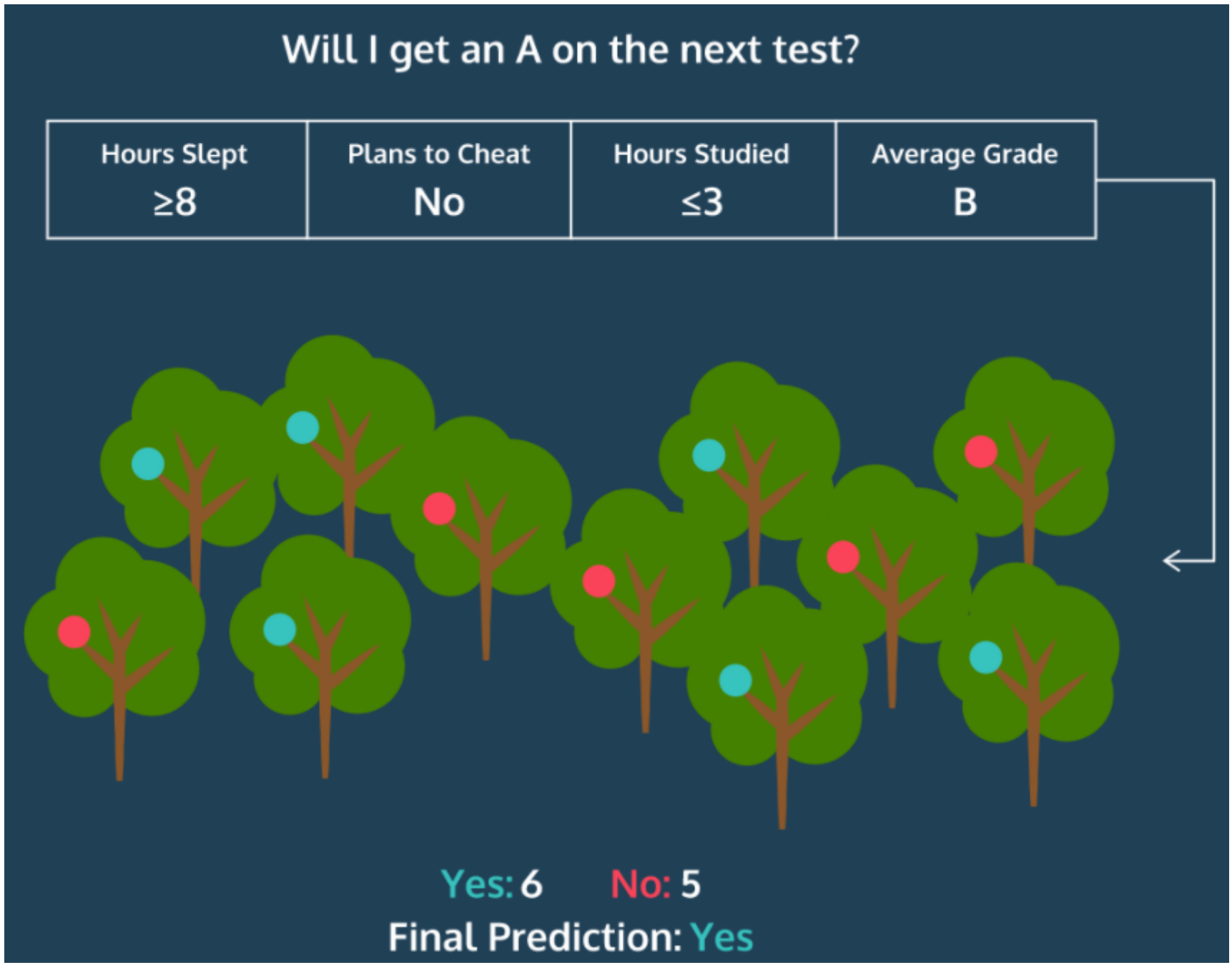
ch7. 앙상블 학습과 랜덤 포레스트

http://hleecaster.com/ml-random-forest-concept/

📖 7.1 투표기반 분류기

< 랜덤포레스트 ft. 결정트리의 오버피팅 한계를 극복하기 위한 전략 >

“무작위 숲”이라는 이름처럼 랜덤 포레스트는 훈련을 통해 구성해 놓은 다수의 나무들로부터 분류 결과를 취합해서 결론을 얻는, 일종의 인기 투표 같은 거다.



물론 몇몇의 나무들이 오버피팅을 보일 순 있지만 다수의 나무를 기반으로 예측하기 때문에 그 영향력이 줄어들게 되어 좋은 일반화 성능을 보인다. 이렇게 좋은 성능을 얻기 위해 다수의 학습 알고리즘을 사용하는 걸 **앙상블(ensemble) 학습법**이라고 부른다.

직접투표 분류기

각 분류기(앙상블)의 예측을 모아서 가장 많이 선택된 클래스를 예측하는 것으로 다수결 투표로 정해지는 분류기를 직접투표 분류기라고 한다. 여기서 분류기는 결정트리 뿐만 아니라 로지스틱 회귀 분류기, SVM 분류기 등 다양한 분류기의 앙상블일 수 있다.

각 분류기가 약한 학습기 (랜덤 추측보다 조금 더 높은 성능을 내는 분류기) 일지라도 충분하게 많고 다양하다면 앙상블은 (높은 정확도를 내는) 강한 학습기가 될 수 있다 → 큰 수의 법칙

앙상블 방법은 예측기가 가능한 서로 독립적일 때 최고의 성능을 발휘한다. 다양한 분류기를 얻는 한가지 방법은 각기 다른 알고리즘으로 학습시키는 것이다. 이렇게 하면 매우 다른 종류의 오차를 만들 가능성이 높아 모델의 정확도를 향상시킨다.

투표 기반 분류기를 만들고 훈련 시키기 & 성능 비교 by 정확도

투표기반 분류기 만들기

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

log_clf = LogisticRegression(solver="lbfgs", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", random_state=42)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')

voting_clf.fit(X_train, y_train)
```

정확도 점수로 성능 살펴보기

```
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))
```

→ 투표기반 분류기가 다른 개별 분류기보다 성능이 조금 더 높게 나옴 (0.912)

간접투표

모든 분류기가 클래스의 확률을 예측할 수 있으면 (즉, predict_proba() 메서드가 있으면) 개별 분류기의 예측을 평균내어 확률이 가장 높은 클래스를 예측하는 방법. 이 방식은 확률이 높은 투표에 비중을 더 두기 때문에 직접투표 방식보다 성능이 높다. 이 방식을 사용하려면 `voting="soft"`를 옵션으로 설정하고 모든 분류기가 클래스의 확률을 추정할 수 있으면 된다. SVC는 기본값에서는 클래스 확률을 제공하지 않으므로 probability 매개변수를 True로 지정해야 한다. 앞선 코드를 간접 투표 방식으로 변경하면 91.2%의 정확도를 얻는다.

📖 7.2 배깅과 페이스팅

다양한 분류기를 만드는 방법

(1) 각기 다른 훈련 알고리즘을 사용 (랜덤포레스트, SVC, 로지스틱 회귀 등등)

(2) 같은 알고리즘을 사용하고 훈련 세트의 서브셋을 무작위로 구성해 같은 분류기를 각기 다르게 학습 시킴 → 배깅,페이스팅

1. 배깅 : 훈련 세트에서 중복을 허용해 샘플링 하는 방식

랜덤 포레스트에서 **각 나무들을 어떻게 생성**하는지 알아야 한다. 결론부터 얘기하면, **배깅(bagging)**이라는 프로세스를 통해 나무를 만든다. 학습 데이터 세트에 총 1000개의 행이 있다고 해보자. 그러면 임의로 100개씩 행을 선택해서 의사결정 트리를 만드는 게 배깅(bagging)이다. 물론 이런 식으로 트리를 만들면 모두 다르겠지만 그래도 어쨌거나 학습 데이터의 일부를 기반으로 생성했다는 게 중요하다. 그리고 이 때 **중복을 허용**해야 한다는 걸 기억하자. 1000개의 행이 있는 가방(bag)에서 임의로 100개 뽑아 첫 번째 트리를 만들고 그 100개의 행은 가방에 도로 집어 넣는다. 그리고 다시 1000개의 행에서 또 임의로 100개를 뽑아 두 번째 트리를 만든 후 다시 가방에 집어 넣고 뭐 이런 식. replacement를 허용.

Bagging Features

여기에 트리를 만들 때 사용될 **속성(feature)들을 제한함**으로써 각 나무들에 **다양성**을 줘야 한다. 원래는 트리를 만들 때 모든 속성들을 살펴보고 정보 획득량이 가장 많은 속성을 선택해서 그걸 기준으로 데이터를 분할했다. 그러나 이제는 **각 분할에서 전체 속성들 중 일부만 고려하여 트리를 작성하도록 하는 전략**이다. 예를 들면 총 25개의 속성이 있는데, 그 중 5개의 속성만 뽑아서 살펴본 후 그 중 정보 획득량이 가장 높은 걸 기준으로 데이터를 분할하는 거다. 그 다음 단계에서도 다시 임의로 5개만 선택해서 살펴보고... 뭐 이런 식. 그렇다면 몇개씩 속성을 뽑는 게 좋을까. 위 예처럼 총 속성이 25개면 5개, 즉 **전체 속성 개수의 제곱근만큼 선택하는 게 가장 좋고**, 경험적으로 그렇게 나타난다고 한다. (일종의 a rule of thumb이다.)

2. 페이스팅 : 중복을 허용하지 않고 샘플링 하는 방식

모든 예측기가 훈련을 마치면 앙상블은 모든 예측기의 예측을 모아 새로운 샘플에 대한 예측을 만든다. **수집 함수**는 전형적으로 분류일 때는 통계적 최빈값 (직접 투표 분류기처럼 가장 많은 예측 결과) 이고 회귀에 대해서는 평균을 계산한다. 개별 예측기는 원본 훈련세트로 훈련시킨 것보다 훨씬 크게 편향되어 있으나, 수집함수를 통과하면 편향과 분산이 모두 감소한다. (일반적으로 편향은 비슷하지만 분산은 줄어드는 경향이 많음)

코드실습

- 페이스팅을 사용하려면 `boostrap=False`로 지정해야 한다. 배깅이 디폴트값!
- `n_jobs` 매개변수는 사이킷런이 훈련과 예측에 사용할 CPU 코어 수를 지정한다. -1로 지정하면 모든 코어를 사용한다.

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

- baggingclassifier는 기반이 되는 분류기가 결정트리처럼 클래스 확률을 추정할 수 있으면 (즉 predict_proba() 함수가 있으면) 직접 투표 대신 자동으로 간접 투표 방식을 사용한다.
- 앙상블은 비슷한 편향에서 더 작은 분산을 만든다. 훈련세트의 오차수가 거의 비슷하지만 결정 경계는 덜 불규칙 하다.
- 부스트래핑은 각 예측기가 학습하는 서브셋에 다양성을 증가시키므로 (중복이 없으니까유) 배깅이 페이스팅보다 편향이 조금 더 높다. 그러나 다양성을 추가하면 예측기들의 상관관계를 줄이므로 분산을 감소시킨다. 전반적으로 배깅이 더 나은 모델을 만들기 때문에 일반적으로 더 선호한다.

oob 평가

배깅을 사용하면 중복을 허용하기 때문에 어떤 샘플은 한 예측기를 위해 여러번 샘플링 되고 어떤 것은 전혀 선택되지 않을 수 있다. 평균적으로 훈련 샘플의 63%정도만 샘플링 되는데, 남은 37%의 샘플을 oob 샘플이라 부른다. 예측기마다 남겨진 oob 샘플은 모두 다르다. 예측기가 훈련되는 동안은 이 샘플을 사용하지 않기 때문에 별도의 검증 세트를 사용하지 않고 oob 샘플을 사용해 평가할 수 있다. 앙상블의 평가는 각 예측기의 oob 평가를 평균 하여 얻는다. 배깅 분류기를 만들 때 `oob_score=True` 을 지정하면 자동으로 훈련이 끝난 후 oob 평가를 수행한다. 평가 점수 결과는 `oob_score_`을 불러오면 된다.

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    oob_score=True, bootstrap=True, n_jobs=-1)
bag_clf.fit(X_train, y_train)
bag_clf.obb_score_
# oob 평가 결과를 보면 이 배깅클래스는 테스트 세트에서 90.1% 정확도를 얻는것으로 보인다.
```

```
from sklearn.metrics import accuracy_score
y_pred = bag_clf.predict(X_test)
accuracy_score(y_test, y_pred)
# 테스트셋에는 91.2% 저오학도로, oob 평가 결과와 비슷하다.
```

oob 샘플에 대한 결정함수의 값을 확인 `oob_decision_function_` 으로 불러올 수 있고, 결정함수는 각 훈련 샘플의 클래스 확률을 반환한다.

```
bag_clf.oob_decision_function_
```

```
array([[0.32275132, 0.67724868],
       [0.34117647, 0.65882353],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.        , 1.        ],
       [0.09497207, 0.90502793],
       [0.31147541, 0.68852459],
```

→ oob 평가는 첫번 째 훈련 샘플이 양성 클래스에 속할 확률을 67.7%로, 음성 클래스에 속할 확률은 32.2%로 추정하고 있다.

📖 7.3 랜덤 패치와 랜덤 서브스페이스

BaggingClassifier는 특성에 대한 샘플링도 지원한다. 샘플링은 `max_features`, `boostrap_features` 두 매개변수로 조절된다. 작동방식은 max_samples, boostrap과 동일하지만 샘플이 아니고 **특성에 대한 샘플링**이다. 따라서 각 예측기는 무작위로 선택한 입력 특성의 일부분으로 훈련된다. 이 기법은 매우 고차원인 데이터 셋 (특히 이미지와 같은 데이터) 을 다룰 때 유용하다.

훈련과 샘플을 모두 샘플링 하는 것을 **랜덤 패치 방식**이라 하고, 훈련 샘플을 모두 사용하고 특성은 샘플링 하는 것을 **랜덤 서브스페이스 방식**이라 한다.

특성 샘플링은 더 다양한 예측기를 만들며 편향을 늘리는 대신 분산을 낮춘다.

📖 7.4 랜덤 포레스트

랜덤 포레스트

일반적으로 배깅 (또는 페이스팅)을 적용한 결정 트리의 앙상블로, max_samples를 훈련 세트의 크기로 지정한다.

코드 실습 : 랜덤 포레스트 훈련

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, random_state=42)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

→ RandomForestClassifier 는 Decision Tree Classifier 의 매개변수와 앙상블 자체를 제어하는데 필요한 BaggingClassifier 매개변수를 모두 가지고 있다.

→ 랜덤 포레스트 알고리즘은 트리의 노드를 분할할 때, 최선의 특성을 찾는 대신 무작위로 선택한 특성 후보중에서 최적의 특성을 찾는 식으로 무작위성을 더 주입한다. 이런 방식은 트리를 다양하게 만들고 편향을 손해보는 대신 분산을 낮추어 전체적으로 더 훌륭한 모델을 만든다.

BaggingClassifier 를 사용해 RandomForestClassifier 와 유사하게 만들어 보기

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(max_features="sqrt", max_leaf_nodes=16),
    n_estimators=500, random_state=42)
```

7.4.1 엑스트라 트리

ㅇ 익스트림 랜덤 트리 (엑스트라 트리) : 트리를 더 무작위하게 만들기 위해 최적의 임곗값을 찾는 대신 후보 특성을 사용해 무작위로 분할한 다음 그 중에서 최상의 분할을 선택하는 극단적으로 무작위한 트리의 랜덤 포레스트

ㅇ 편향이 늘어나지만 대신 분산을 낮추게 된다. 최적의 임곗값을 찾지 않아 알고리즘에서 시간이 많이 단축되어 일반적인 랜덤 포레스트보다 엑스트라 트리가 훨씬 빠르다.

ㅇ 사이킷런의 `ExtraTreesClassifier`를 사용함

7.4.2 특성 중요도

랜덤 포레스트는 특성의 상대적인 중요도를 측정하기 쉽다. 사이킷런은 어떤 특성을 사용한 노드가 평균적으로 불순도를 얼마나 감소시키는지 확인하여 특성의 중요도를 측정한다. 정확히 말하면 가중치 평균이며 각 노드의 가중치는 연관된 훈련 샘플의 수와 같다. 훈련이 끝난뒤 사이킷런은 특성마다 자동으로 이 점수를 계산하고 중요도의 전체 합이 1이 되도록 결과값을 정규화해서 보여준다. `feature_importances_` 변수에 저장되어 있다.

(붓꽃예제)

```
from sklearn.datasets import load_iris
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, random_state=42)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
    print(name, score)
```

```
rnd_clf.feature_importances_
```

📖 7.5 부스팅

부스팅

약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 앙상블 기법으로, 앞의 모델을 보완해 나가면서 일련의 예측기를 학습 시키는 방법이다. 여러 부스팅 방법이 있지만 가장 인기 있는 것은 **에이다부스트**와 **그레디언트 부스팅**이 있다.

에이다 부스트

ㅇ 이전 예측기를 보완하기 위해, 이전 모델이 과소적합했던 훈련샘플의 가중치를 더 높이게 하여, 새로운 예측기가 학습하기 어려운 샘플에 점점 더 맞춰지게 만드는 방식

ㅇ 약한 분류기(weak classifier)들은 한 번에 하나씩 순차적으로 학습을 진행한다. 먼저 학습된 분류기는 제대로 분류를 해내는 데이터와 제대로 분류해내지 못하는 데이터들이 발생한다. 먼저 학습된 분류기가 제대로 분류한 결과 정보와 잘못 분류한 결과 정보를 다음 분류기에 전달한다. 다음 분류기는 이전 분류기로부터 받은 정보를 활용하여 잘 분류해내지 못한 데이터들의 가중치(weight)를 높인다. 즉, **이전 분류기가 잘못 분류한 샘플의 가중치를 adaptive하게 바꿔가며 잘 못 분류되는 데이터에 더 집중하여 학습이 더 잘되게 한다.** 이러한 특징 때문에 adaptive라는 이름이 붙음. 최종 분류기(strong classifier)는 이전에 학습한 약한 분류기들에 각각 가중치를 적용하고 조합하여 학습을 진행한다. 정리하면 예측 성능이 낮은 약한 분류기들을 조합하여 최종적으로 조금 더 성능이 좋은 강한 분류기 하나를 만드는 것이다. 약한 분류기들이 상호보완적(adaptive)으로 학습해나가고, 이러한 약한 분류기들을 조합하여 하나의 분류기를 만들기 때문에 boosting이 된다.

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
```

```
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

그레디언트 부스팅

에이다부스트처럼 반복마다 샘플의 가정치를 수정하는 대신 이전 예측기가 만든 **잔여오차**에 새로운 예측기를 학습시키는 방법

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 된다. 이러한 방법을 계속하면 잔차는 계속해서 줄어들게되고, training set을 잘 설명하는 예측 모형을 만들 수 있게 된다. 하지만 이러한 방식은 bias는 상당히 줄일 수 있어도, 과적합이 일어날 수도 있다는 단점이 있다.

```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0, random_state=42)
gbrt.fit(X, y)
```

최적화된 그레디언트 부스팅 구현 라이브러리 : xgboost

```
import xgboost
xgb_reg = xgboost.XGBRegressor(random_state=42)
xgb_reg.fit(X_train, y_train)
y_pred = xgb_reg.predict(X_val)


# 자동 조기종료가 있는 버전
xgb_reg.fit(X_train, y_train,
                eval_set=[(X_val, y_val)], early_stopping_rounds=2)
y_pred = xgb_reg.predict(X_val)
```

📖 7.6 스태킹

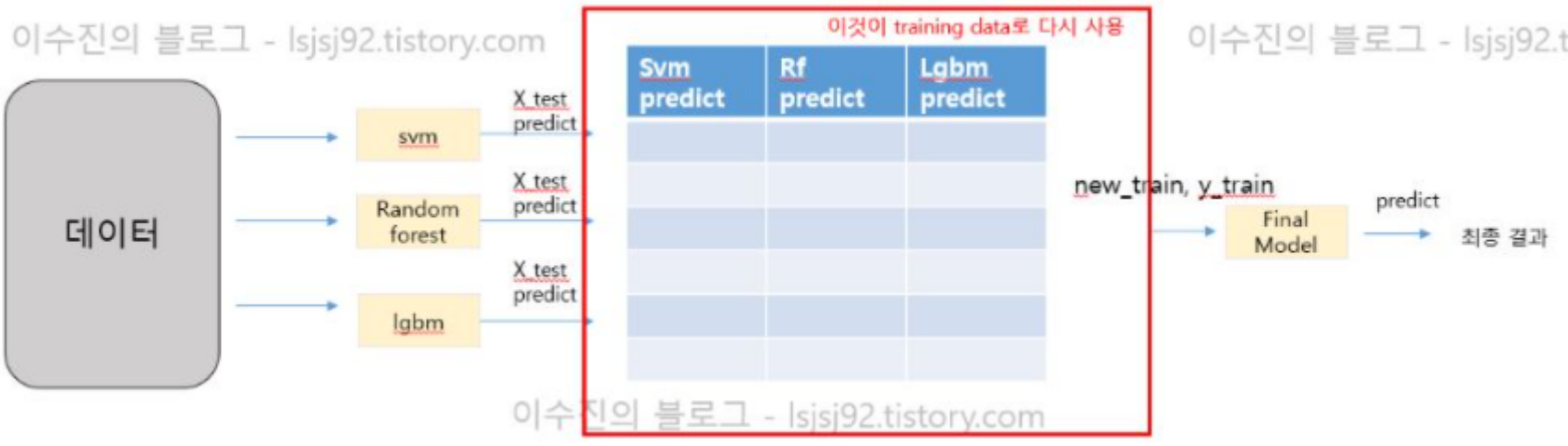
스태킹

"앙상블에 소한 모든 예측기의 예측을 취합하는 '함수'를 사용하는 대신 취합하는 '모델'을 만들자"는 아이디어에서 출발한다. 마지막 예측기(블렌더 또는 메타 학습기라 부름)가 앙상블 예측기의 결과들을 수집해 최종 예측을 만드는 방식을 말한다. 마지막 예측기는 이전의 개별 모델이 '예측했던 데이터'를 다시 training set으로 받아들여 사용해 학습하여 결과를 도출한다.

따라서 stacking ensemble을 할 때는 2가지 개념의 모델이 필요합니다.

1. 개별 모델들(복수)

2. 최종 모델(하나)


이수진의 블로그 - lsjsj92.tistory.com
이것이 training data로 다시 사용
이수진의 블로그 - lsjsj92.t
데이터
svm
Random forest
lgbm
X test predict
X test predict
X test predict
Svm predict
Rf predict
Lgbm predict
new_train, y_train
Final Model
predict
최종 결과
이수진의 블로그 - lsjsj92.tistory.com