Ensemble Learning and Random Forests

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

국립한경대학교
HANKYONG NATIONAL UNIV.

어려운 질문을 수천명의 사람들에게 던져서 그들의 답을 모아서 합계를 낸다고 해보자.

⇒ 많은 경우에 이렇게 얻은 답이 전문가의 답보다 더 좋다.

⇒ **"군중의 지혜" 라고 함**

마찬가지로, predictor들의 그룹에 의해 수행된 예측을 모아서 합계를 낸다면 최상의 개별 predictor가 예측한 것보다 더 좋을 수 있다:

⇒ Predictor들의 그룹을 "앙상블(ensemble)"이라 함

⇒ 이 기법을 Ensemble Learning이라 함

⇒ Ensemble Learning 알고리즘을 Ensemble method라 함

예) Ensemble method의 예

- 각 Decision Tree classifier가 training set의 서로 다른 랜덤 서브 세트에 기반하여 Decision Tree classifier들의 그룹을 학습시킬 수 있다.
- 예측을 위해, 개별 트리의 예측을 모두 얻어서 가장 많은 득표를 한 클래스를 예측한다.
- 이런 Decision Tree의 ensemble을 **Random Forest**(현존하는 가장 강력한 ML 알고리즘)라 함

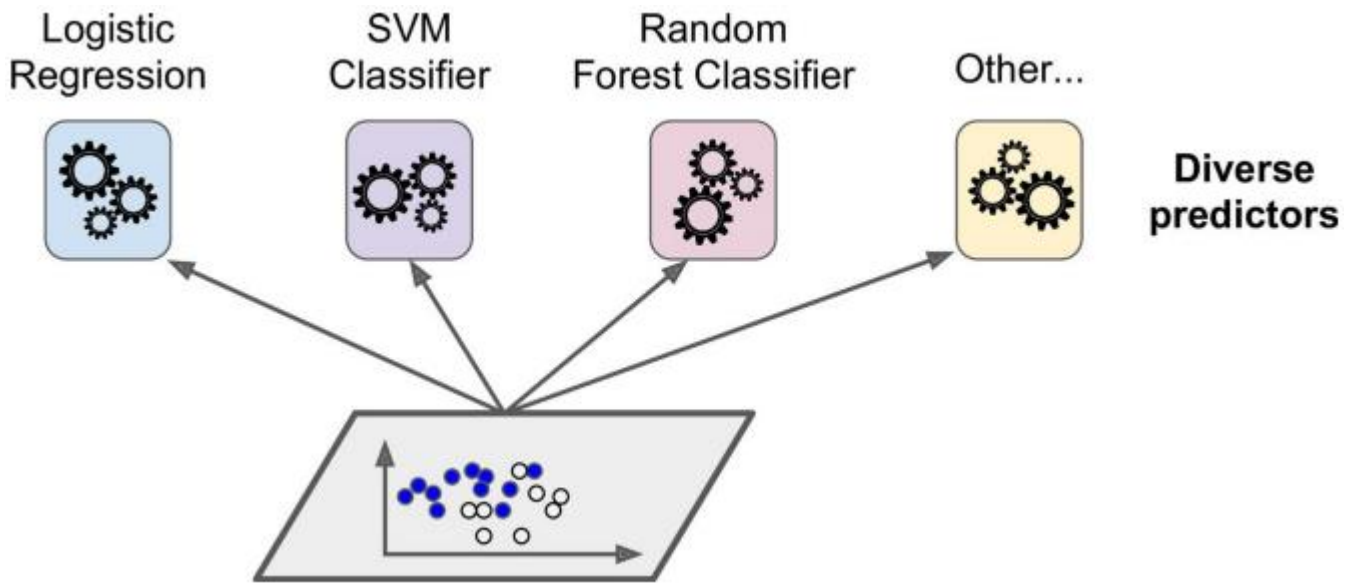
Ensemble method는 보통 프로젝트 마지막 단계에서 이용한다.

- 프로젝트 마지막 단계에서는 이미 몇몇 좋은 predictor들을 구축했을 것이고 그들을 훨씬 더 좋은 predictor로 조합할 수 있을 것이다.
- 사실, 유명 Machine Learning 경시대회에서 우승한 솔루션들을 보면 많은 경우에 Ensemble method를 이용하고 있다. (http://netflixprize.com/)

Voting Classifiers

몇 개의 classifier를 학습시켰고, 각 classifier는 약 80% 정도의 정확도(accuracy)를 달성했다고 하자.

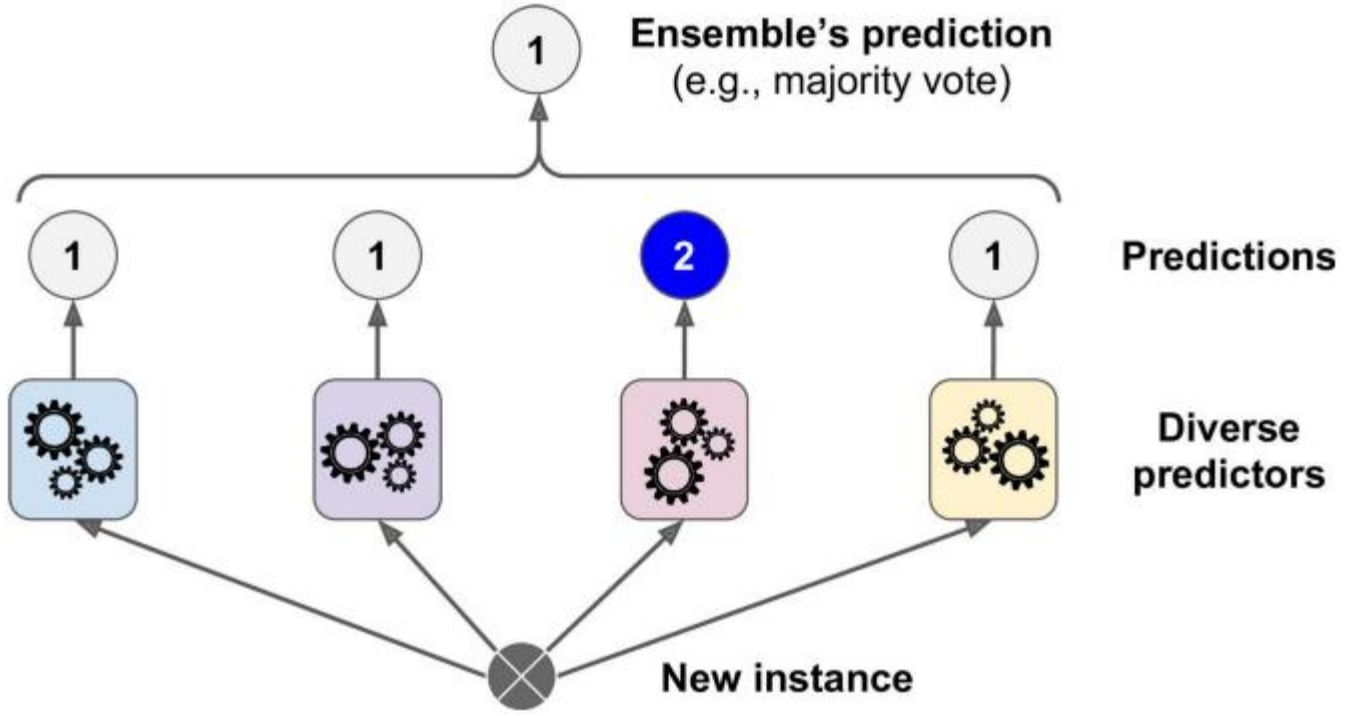
- Logistic Regression classifier, an SVM classifier, a Random Forest classifier, and perhaps a few more.



Voting Classifiers

훨씬 더 좋은 classifier를 만들기 위한 매우 간단한 방법:

- 각 classifier가 예측한 결과를 모두 모아서 득표수가 가장 많은 클래스를 예측
- 이런 majority-vote classifier를 "**hard voting**" classifier라 함
- Hard voting classifier predictions:



Voting Classifiers

놀랍게도, voting classifier가 앙상블에 있는 최상의 classifier보다 더 높은 정확도를 달성한다.

- 각 classifier가 **weak learner**라 할지라도 충분한 개수가 주어진다면 **strong learner**가 될 수 있다.

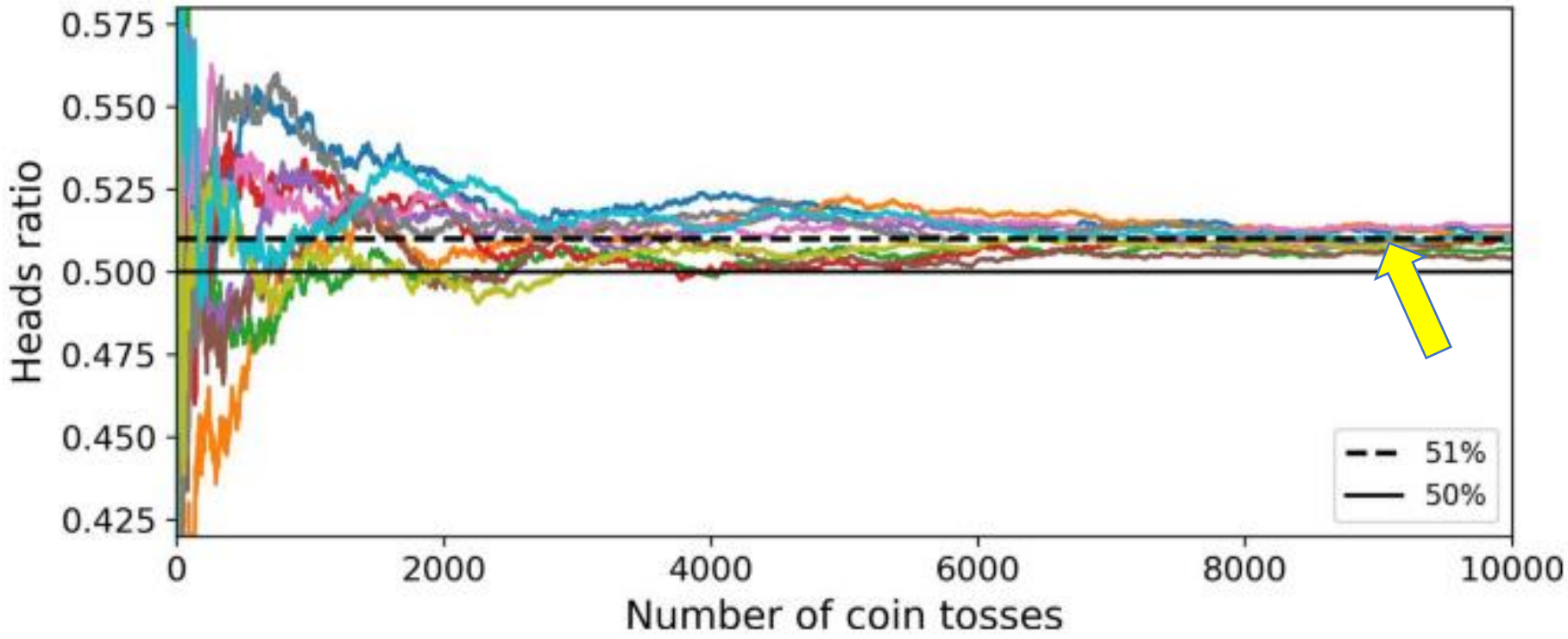
How is this possible?

유사 예: 앞면이 나올 확률이 51%, 뒷면이 나올 확률이 49%인 편향된 동전이 있다고 하자.

- 1,000회를 던진다면 보통은 대략 앞면 510회, 뒷면 490회 정도 나온다.
- 통계 이론상 1,000번 던져서 앞면이 더 많이 나올 확률은 약 75% 정도
- 동전을 더 많이 던질수록 앞면이 더 많이 나올 확률은 높아짐 (10,000번 정도 던지면 97%)
- => 대수의 법칙: 동전을 계속 던진다면 앞면이 나올 비율이 앞면의 확률(51%)에 점점 더 가까이 간다.

10개의 동전을 10,000번 던지는 실험:

- 궁극적으로 10개 시리즈 모두 51%에 수렴한다:



Voting Classifiers

개별적으로는 51%만 정확한 1,000개의 classifier를 포함하는 앙상블을 만든다고 하자:

- 만일 다수 표를 얻는 클래스로 예측을 결정한다고 하면 75%의 정확도까지 얻을 수 있다!
- => 그러나 이것은 모든 classifier가 완전하게 독립적일 경우에 한하여 사실이다.
- => 따라서 classifier를 동일 데이터에 기반하여 학습시킨다면 확실히 앙상블의 정확도가 낮아질 것이다.

앙상블 메소드 => predictor들이 가급적 서로 독립적일 때 최상으로 동작한다.

- **다양한 classifier**를 얻기 위한 한 가지 방법 => 서로 다른 알고리즘을 이용하여 학습시키는 것

Voting Classifiers

Scikit-Learn의 voting classifier 생성, 학습 (3개의 classifier로 구성, training set: moons dataset):

```
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)
```

Test set에 대한 각 classifier의 정확도:

```
>>> 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
```

← Voting classifier가 약간 더 좋다!

Voting Classifiers

Soft voting:

- 모든 classifier가 클래스 확률을 추정할 수 있다면 (`predict_proba()`를 가진다면)

  ⇒ Scikit-Learn에게 가장 높은 확률을 가진 클래스를 예측하라고 말해 줄 수 있다.

  ⇒ "soft voting"이라 함

- voting="hard" => voting="soft"로 변경하기만 하면 됨

  (단, 모든 classifier가 클래스 확률을 추정할 수 있어야 한다.)

- SVC => 디폴트로 클래스 확률을 추정할 수 없음 (cross-validation을 사용해야 함)
- 앞 페이지의 예제를 soft voting으로 변경하면 91.2% 이상의 정확도를 얻을 수 있음

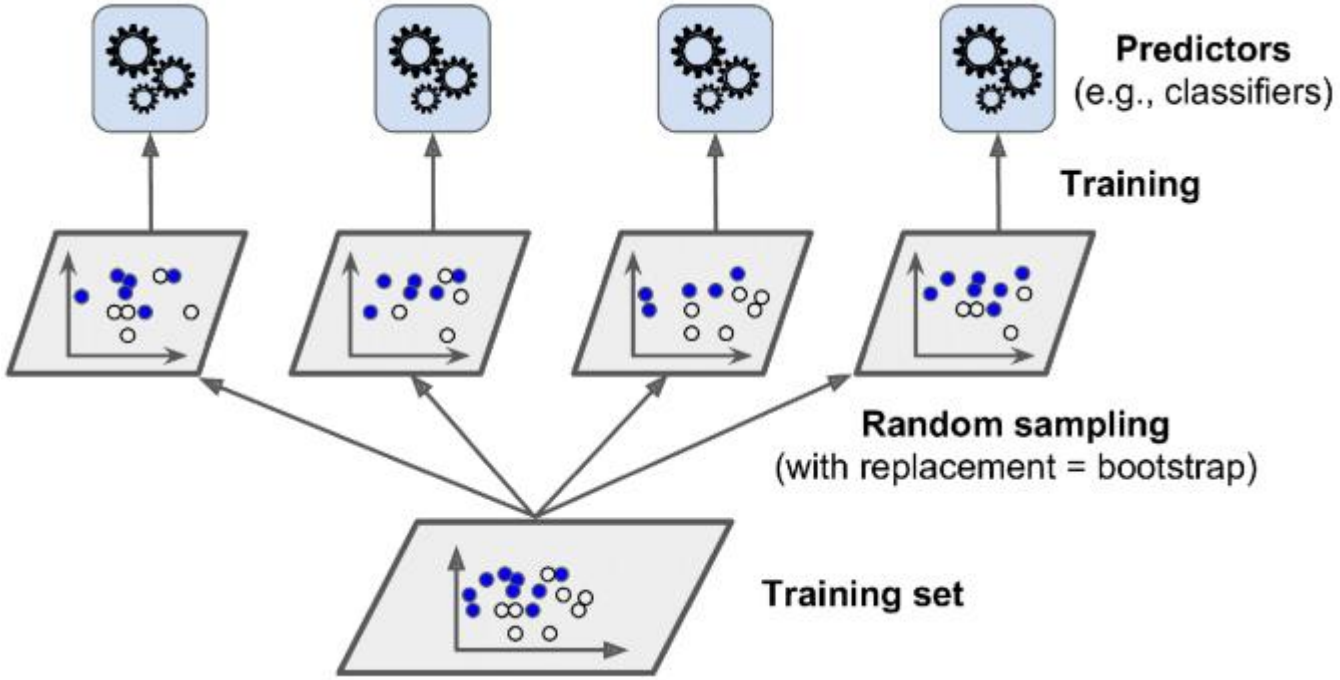
Bagging and Pasting

다양한 classifier를 얻기 위한 또 다른 방법:

- 모든 predictor를 training set의 서로 다른 랜덤 샘플 세트에 대해 동일한 학습 알고리즘으로 학습시킨다.
- 샘플링이 복원추출(with replacement) 방식으로 수행되면 "**bagging**" 방법이라 함
- 샘플링이 비복원추출(without replacement) 방식으로 수행되면 "**pasting**" 방법이라 함

Bagging and Pasting

Bagging과 Pasting => training set의 서로 다른 랜덤 샘플들에 대해 몇 개의 predictor를 학습시킨다:



- 모든 predictor를 학습시켰다면 모든 predictor들의 예측을 모음으로써 새 인스턴스를 예측할 수 있다.

Bagging and Pasting in Scikit-Learn

Scikit-Learn => bagging과 pasting 둘 다를 위한 `BaggingClassifier`라는 API 지원

- 500개의 Decision Tree classifier로 구성된 앙상블
- 각 classifier => 복원추출로 랜덤하게 샘플링된 100개의 인스턴스 세트 상에서 학습시킴
- Pasting을 사용하고 싶다면 `bootstrap=False`
- `n_jobs` => Scikit-Learn에게 사용할 CPU 코어 개수를 알려 줌 (-1: 이용 가능한 모든 코어 사용)

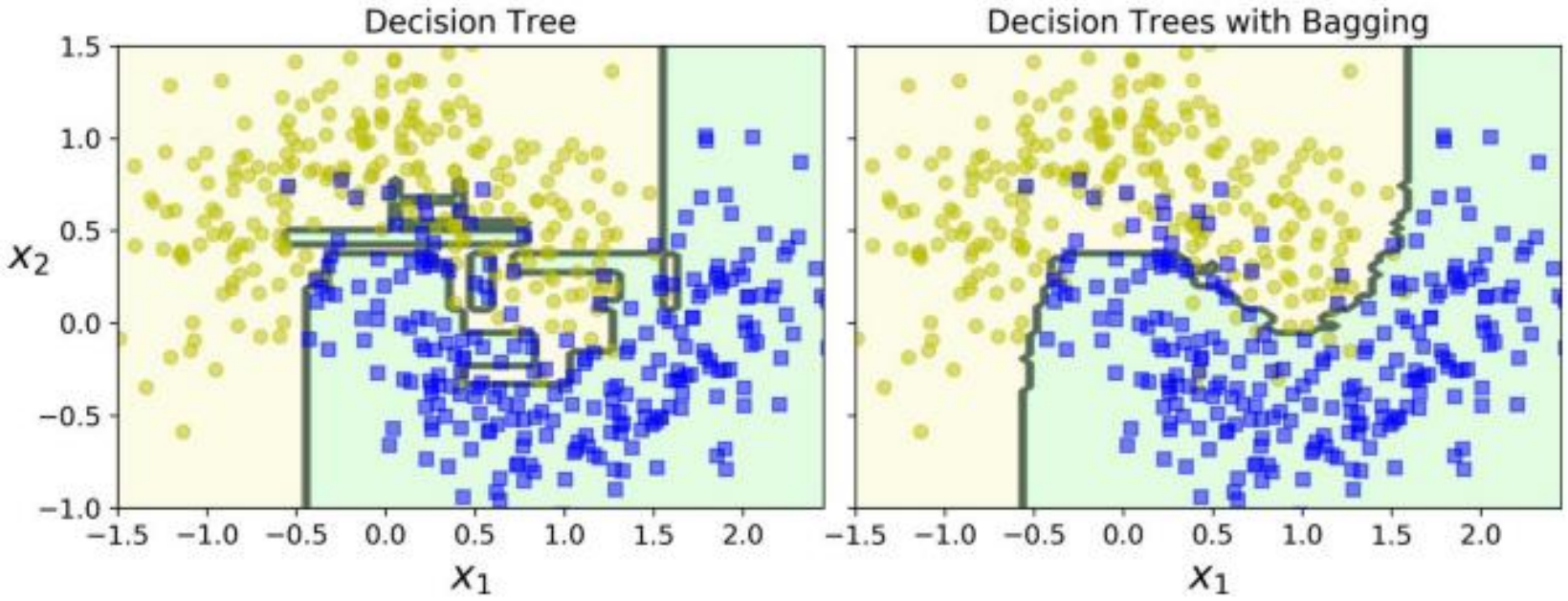
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

- `BaggingClassifier` => 자동으로 soft voting 수행 (클래스 확률을 추정할 수 있다면)

Bagging and Pasting in Scikit-Learn

1개 Decision Tree의 decision boundary vs. 500개 트리의 bagging 앙상블 트리의 decision boundary

- Moons dataset에 대해 학습시킴
- 앙상블의 예측이 1개 Decision Tree보다 일반화 성능이 훨씬 더 좋아 보임



Out-of-Bag Evaluation

Bagging 방법:

- 어떤 인스턴스들은 주어진 predictor에 대해 여러 번 샘플링 될 수 있음
- 또 다른 인스턴스들은 전혀 샘플링 되지 않을 수 있음

`BaggingClassifier`

- 복원추출(bootstrap=True)로 `m`(training set size)개의 인스턴스 샘플링
- 각 predictor에 대해 63%만이 샘플링 됨
- 나머지 37%는 샘플링 되지 않음 => *`out-of-bag`* (oob) 인스턴스 세트라 함
- Scikit-Learn => `BaggingClassifier` 생성 시 학습 후 자동으로 oob 평가가 이루어지도록 `oob_score=True`로 설정 가능

Out-of-Bag Evaluation

자동 oob 평가 결과(evaluation score):

```
>>> bag_clf = BaggingClassifier(
...     DecisionTreeClassifier(), n_estimators=500,
...     bootstrap=True, n_jobs=-1, oob_score=True)
...
>>> bag_clf.fit(X_train, y_train)
>>> bag_clf.oob_score_
0.9013333333333332
```

실제 test set에 대해 평가한 정확도:

```
>>> from sklearn.metrics import accuracy_score
>>> y_pred = bag_clf.predict(X_test)
>>> accuracy_score(y_test, y_pred)
0.9120000000000003
```

91.2%: 충분히 가깝다!

Random Patches

`BaggingClassifier`

- 입력 특성 또한 샘플링 할 수 있게 지원
- 샘플링 => 2개의 hyperparameter에 의해 지원: **max_features**, **bootstrap_features**
- max_samples, bootstrap과 같은 방식으로 동작하지만, instance 샘플링이 아니라 **feature** 샘플링 수행
- 각 predictor는 입력 특성들의 랜덤 서브 세트에 대해 학습 됨
- 이미지와 같은 고차원 입력을 다루어야 할 때 유용
- Instances와 features 둘 다를 샘플링 하는 방식을 **Random Patches** 방식이라 함

Random Forests

Random Forests

- Decision Tree의 앙상블, 보통 bagging 방식으로 학습시킴 (`max_samples = training_set_size`)
- `BaggingClassifier`를 구축하여 `DecisionTreeClassifier`에 전달하는 것이 아니라, `RandomForestClassifier` 클래스 사용
- `RandomForestClassifier` 클래스 => 더 편리하고 Decision Tree에 최적화 되어 있음
- 500개 트리로 구성된 Random Forest classifier를 학습시키기 위해 가능한 모든 코어 사용:

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

- `RandomForestRegressor`도 있음

Random Forest 알고리즘 => 트리를 키워 나갈 때 extra randomness 도입

- 앞 `RandomForestClassifier`와 동등한 `BaggingClassifier`:

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16),
    n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1)
```

Randomness를 극대화 시킨 random 트리들의 Forest:

- Extremely Randomized Trees 앙상블이라 함 (간단히 Extra-Trees)
- 정식 Random Forests보다 훨씬 더 빠르다.
- Scikit-Learn의 `ExtraTreesClassifier` 클래스를 이용하여 Extra-Trees classifier를 생성할 수 있음

Feature Importance

Random Forests의 또 다른 위대한 특징:

- 각 특성의 상대적 중요도를 측정 가능
- Scikit-Learn의 특성 중요도 측정 방식:

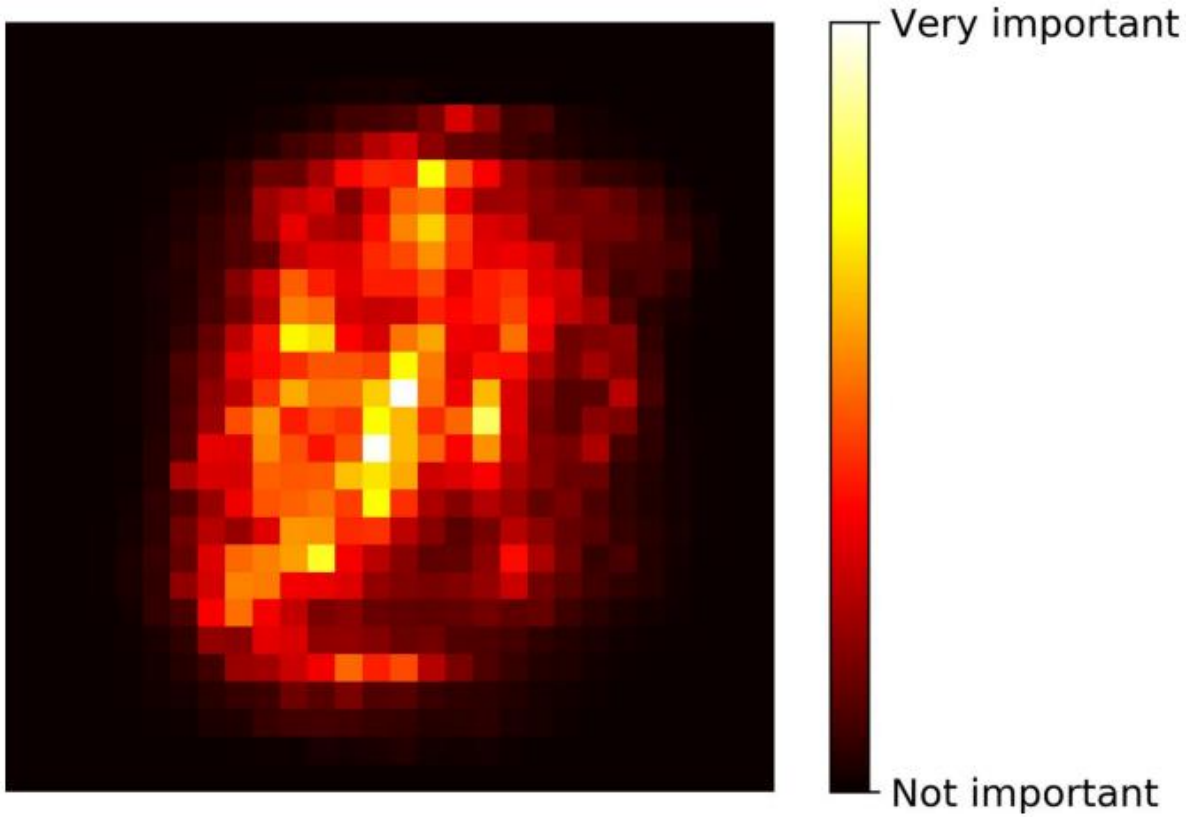
  ⇒ 해당 특성을 사용하는 트리 노드가 불순도(impurity)를 얼마나 줄이는가를 측정
- Scikit-Learn => 학습 후 자동으로 점수 계산 => 모든 중요도의 합이 1이 되도록 스케일 조정
- `feature_importances_` 변수로 접근 가능

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
>>> rnd_clf.fit(iris["data"], iris["target"])
>>> for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
...     print(name, score)
...
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Feature Importance

MNIST dataset에 대해 학습된 Random Forest classifier:

- 각 픽셀의 중요도를 PLOT!
- MNIST pixel importance (according to a Random Forest classifier)



실습과제 10-1

본문에 나오는 전체 내용 PyCharm에서 실행하기

참고: 제 10강 실습과제 10-1 Ensemble Learning and Random Forests.pdf

실습과제 10-2

Load the MNIST data, and split it into a training set, a validation set, and a test set (e.g., use 50,000 instances for training, 10,000 for validation, and 10,000 for testing). Then train various classifiers, such as a Random Forest classifier, an Extra-Trees classifier, and an SVM classifier. Next, try to combine them into an ensemble that outperforms each individual classifier on the validation set, using soft or hard voting. Once you have found one, try it on the test set. How much better does it perform compared to the individual classifiers?

참고: 제 10강 실습과제 10-2 Voting Classifier - MNIST dataset.pdf