앙상블(Ensemble)

- 일반화와 강건성(Robustness)을 향상시키기 위해 여러 모델의 예측 값을 결합하는 방법
- 앙상블에는 크게 두가지 종류가 존재
  - 평균 방법
    - 여러개의 추정값을 독립적으로 구한뒤 평균을 취함
    - 결합 추정값은 분산이 줄어들기 때문에 단일 추정값보다 좋은 성능을 보임
  - 부스팅 방법
    - 순차적으로 모델 생성
    - 결합된 모델의 편향을 감소 시키기 위해 노력
    - 부스팅 방법의 목표는 여러개의 약한 모델들을 결합해 하나의 강력한 앙상블 모델을 구축하는 것

Bagging meta-estimator

- bagging은 bootstrap aggregating의 줄임말
- 원래 훈련 데이터셋의 일부를 사용해 여러 모델을 훈련
- 각각의 결과를 결합해 최종 결과를 생성
- 분산을 줄이고 과적합을 막음
- 강력하고 복잡한 모델에서 잘 동작

```
from sklearn.datasets import load_iris, load_wine, load_breast_cancer
from sklearn.datasets import load_boston, load_diabetes
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_validate
```

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import BaggingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
```

Bagging을 사용한 분류

데이터셋 불러오기

```
iris = load_iris()
wine = load_wine()
cancer = load_breast_cancer()
```

KNN

붓꽃 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    KNeighborsClassifier()
)

bagging_model = BaggingClassifier(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0017046451568603516 (+/- 0.0015271525312839148)
avg score time: 0.0018569469451904298 (+/- 0.00026105632928522313)
avg test score: 0.96 (+/- 0.024944382578492935)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.019382762908935546 (+/- 0.004653281278427905)
avg score time: 0.007428598403930664 (+/- 0.0006380455258440546)
avg test score: 0.96 (+/- 0.024944382578492935)
```

와인 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    KNeighborsClassifier()
)
```

```
bagging_model = BaggingClassifier(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0018463611602783203 (+/- 0.0004702954604229703)
avg score time: 0.002838754653930664 (+/- 0.0008537944963933942)
avg test score: 0.9493650793650794 (+/- 0.037910929811115976)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.018559646606445313 (+/- 0.0031639743130724763)
avg score time: 0.007317113876342774 (+/- 0.0002694194674769198)
avg test score: 0.9385714285714286 (+/- 0.0405666589341837)
```

유방암 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    KNeighborsClassifier()
)

bagging_model = BaggingClassifier(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=cancer.data, y=cancer.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0026380538940429686 (+/- 0.0007179755221430953)
avg score time: 0.009376621246337891 (+/- 0.0021607756659078035)
avg test score: 0.9648501785437045 (+/- 0.009609970350036127)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
```

```
    X=cancer.data, y=cancer.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.021631526947021484 (+/- 0.002422563814047447)
avg score time: 0.015113210678100586 (+/- 0.0008714074617950579)
avg test score: 0.9666200900481291 (+/- 0.006514994412359255)
```

SVC

붓꽃 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    SVC()
)

bagging_model = BaggingClassifier(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0015958309173583984 (+/- 0.0004761209736233793)
avg score time: 0.0005391120910644531 (+/- 0.00016173873582662102)
avg test score: 0.9666666666666666 (+/- 0.02108185106778919)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.030159521102905273 (+/- 0.0066852049400411805)
avg score time: 0.0031513690948486326 (+/- 0.00017335948541082044)
avg test score: 0.9533333333333334 (+/- 0.03399346342395189)
```

와인 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    SVC()
)

bagging_model = BaggingClassifier(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0030254364013671876 (+/- 0.0008948659224917901)
avg score time: 0.0007365703582763672 (+/- 0.00015224780987523393)
avg test score: 0.9833333333333334 (+/- 0.022222222222222233)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.02991485595703125 (+/- 0.00454567469115838)
avg score time: 0.003741741180419922 (+/- 0.0006888918755317299)
avg test score: 0.9438095238095239 (+/- 0.02521462420645036)
```

유방암 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    SVC()
)

bagging_model = BaggingClassifier(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=cancer.data, y=cancer.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.006421232223510742 (+/- 0.0009341753701631633)
```

```
avg score time: 0.001578044891357422 (+/- 0.0003029881387645502)
avg test score: 0.9736376339077782 (+/- 0.014678541667933545)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=cancer.data, y=cancer.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.03567805290222168 (+/- 0.002532612681952705)
avg score time: 0.0068646430969238285 (+/- 0.0006457386226452189)
avg test score: 0.9701288619779538 (+/- 0.01188544695418272)
```

Decision Tree

붓꽃 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    DecisionTreeClassifier()
)

bagging_model = BaggingClassifier(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0018978118896484375 (+/- 0.0004923319066057832)
avg score time: 0.001008749008178711 (+/- 0.0006182482554894767)
avg test score: 0.9666666666666668 (+/- 0.036514837167011066)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.027107906341552735 (+/- 0.003940641173917783)
avg score time: 0.0030245304107666014 (+/- 0.0010553476651669088)
```

```
avg test score: 0.9466666666666667 (+/- 0.04521553322083511)
```

▼ 와인 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    DecisionTreeClassifier()
)

bagging_model = BaggingClassifier(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.003184080123901367 (+/- 0.002313532025900752)
avg score time: 0.0006137847900390625 (+/- 0.00022403845647841897)
avg test score: 0.8876190476190475 (+/- 0.04989432016453655)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0254638671875 (+/- 0.0034064851987652936)
avg score time: 0.0023194313049316405 (+/- 0.0001618149970121824)
avg test score: 0.95 (+/- 0.061864048475889125)
```

▼ 유방암 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    DecisionTreeClassifier()
)

bagging_model = BaggingClassifier(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=cancer.data, y=cancer.target,
    cv=5
)
```

```
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.009707307815551758 (+/- 0.0006851004330212195)
avg score time: 0.0007386207580566406 (+/- 7.914850484358001e-05)
avg test score: 0.9103400093153237 (+/- 0.023271691012174994)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=cancer.data, y=cancer.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.03433384895324707 (+/- 0.0021893576420278153)
avg score time: 0.0025305747985839844 (+/- 0.00027949032202823423)
avg test score: 0.9542928116752059 (+/- 0.016120893694892183)
```

Bagging을 사용한 회귀

데이터셋 불러오기

```
boston = load_boston()
diabetes = load_diabetes()
```

KNN

보스턴 주택 가격 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor()
)

bagging_model = BaggingRegressor(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.001988840103149414 (+/- 0.0008370103626557836)
avg score time: 0.0023296833038330077 (+/- 0.0010180853070540297)
avg test score: 0.47357748833823543 (+/- 0.13243123464477455)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0179995059967041 (+/- 0.002663403263652928)
avg score time: 0.009055328369140626 (+/- 0.0005249510734832513)
avg test score: 0.4631017843797302 (+/- 0.07262067760414836)
```

당뇨병 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor()
)

bagging_model = BaggingRegressor(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=diabetes.data, y=diabetes.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0029625892639160156 (+/- 0.003495719806407071)
avg score time: 0.002025938034057617 (+/- 0.0004821899993836502)
avg test score: 0.3689720650295623 (+/- 0.044659049060165365)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=diabetes.data, y=diabetes.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.019009971618652345 (+/- 0.0043604183449507195)
avg score time: 0.00879979133605957 (+/- 0.0003814429035293206 5)
avg test score: 0.42493605915338345 (+/- 0.043176561923537804)
```

SVR

보스턴 주택 가격 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    SVR()
)

bagging_model = BaggingRegressor(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std()
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.01580209732055664 (+/- 0.0015818938324450066)
avg score time: 0.0023703575134277344 (+/- 4.6605801371816316e-05)
avg test score: 0.17631266230186618 (+/- 0.5224914915128981)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std()
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0528831958770752 (+/- 0.004375445165611861)
avg score time: 0.008875465393066407 (+/- 0.00023623701511516984)
avg test score: 0.10893486971139148 (+/- 0.34305655896486204)
```

당뇨병 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    SVR()
)

bagging_model = BaggingRegressor(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=diabetes.data, y=diabetes.target,
```

```
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.011080646514892578 (+/- 0.0017639090948212458)
avg score time: 0.0021448612213134767 (+/- 0.0003266781894649412)
avg test score: 0.14659936199629434 (+/- 0.02190798003342928)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=diabetes.data, y=diabetes.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.042175579071044925 (+/- 0.006007041635110622)
avg score time: 0.007076787948608399 (+/- 0.0005758847346321918)
avg test score: 0.05950392521980994 (+/- 0.040345997608972006)
```

Decision Tree

보스턴 주택 가격 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    DecisionTreeRegressor()
)

bagging_model = BaggingRegressor(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.005176115036010742 (+/- 0.0017838484933113063)
avg score time: 0.0007643699645996094 (+/- 8.940347028887446e-05)
avg test score: 0.12219609955826476 (+/- 0.7977201829143907)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=boston.data, y=boston.target,
    cv=5
```

```
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std()))
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.029555654525756835 (+/- 0.0029404988277694738)
avg score time: 0.0024316787719726564 (+/- 7.152102773842777e-05)
avg test score: 0.4635945149023241 (+/- 0.28418078936603136)
```

▾ 당뇨병 데이터

```
base_model = make_pipeline(
    StandardScaler(),
    DecisionTreeRegressor()
)

bagging_model = BaggingRegressor(base_model, n_estimators=10, max_samples=0.5, max_features=0.5)
```

```
cross_val = cross_validate(
    estimator=base_model,
    X=diabetes.data, y=diabetes.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std()))
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.004505157470703125 (+/- 0.001821271601341477)
avg score time: 0.0008062839508056641 (+/- 0.00016820875313370263)
avg test score: -0.16365550900077724 (+/- 0.08585754671709693)
```

```
cross_val = cross_validate(
    estimator=bagging_model,
    X=diabetes.data, y=diabetes.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std()))
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.03084864616394043 (+/- 0.0021574839005306613)
avg score time: 0.0024981975555419924 (+/- 0.0001822656879560538 6)
avg test score: 0.398474945930005 (+/- 0.08409613626437067)
```

Forests of randomized trees

- `sklearn.ensemble` 모듈에는 무작위 결정 트리를 기반으로하는 두 개의 평균화 알고리즘이 존재
    - Random Forest
    - Extra-Trees

- 모델 구성에 임의성을 추가해 다양한 모델 집합이 생성
- 앙상블 모델의 예측은 각 모델의 평균

```
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
```

Random Forests 분류

```
model = make_pipeline(
    StandardScaler(),
    RandomForestClassifier()
)
```

```
cross_val = cross_validate(
    estimator=model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.12527718544006347 (+/- 0.004017817571142579)
avg score time: 0.007357358932495117 (+/- 9.653038008925902e-05)
avg test score: 0.9666666666666668 (+/- 0.02108185106778919)
```

```
cross_val = cross_validate(
    estimator=model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.13404903411865235 (+/- 0.009162142425117537)
avg score time: 0.009136962890625 (+/- 0.0020595115558816397)
avg test score: 0.9776190476190475 (+/- 0.020831783767013237)
```

```
cross_val = cross_validate(
    estimator=model,
    X=cancer.data, y=cancer.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.18993163108825684 (+/- 0.005235385858728354)
```

```
avg score time: 0.008118677139282226 (+/- 0.0001124074601071386)
avg test score: 0.9648812296227295 (+/- 0.025402107570993933)
```

Random Forests 회귀

```
model = make_pipeline(
    StandardScaler(),
    RandomForestRegressor()
)
```

```
cross_val = cross_validate(
    estimator=model,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std()
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-e27e11821c68> in <module>()
      1 cross_val = cross_validate(
      2     estimator=model,
----> 3     X=boston.data, y=boston.target,
      4     cv=5
      5 )

NameError: name 'boston' is not defined
```

SEARCH STACK OVERFLOW

```
cross_val = cross_validate(
    estimator=model,
    X=diabetes.data, y=diabetes.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std()
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

Extremely Randomized Trees 분류

```
            2     estimator=model,
```

```
model = make_pipeline(
    StandardScaler(),
    ExtraTreesClassifier()
)
```

```
cross_val = cross_validate(
    estimator=model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.09486193656921386 (+/- 0.009817572434588041)
avg score time: 0.008888006210327148 (+/- 0.0014334307130221314)
avg test score: 0.9533333333333334 (+/- 0.03399346342395189)
```

```
cross_val = cross_validate(
    estimator=model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.09533910751342774 (+/- 0.0036298334956324746)
avg score time: 0.008881759643554688 (+/- 0.0012660765527540971)
avg test score: 0.9722222222222221 (+/- 0.024845199749997673)
```

```
cross_val = cross_validate(
    estimator=model,
    X=cancer.data, y=cancer.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.11192851066589356 (+/- 0.0038486473180337397)
avg score time: 0.00907435417175293 (+/- 0.00010430169494407668)
avg test score: 0.9666045645086166 (+/- 0.01164730071870122)
```

Extremely Randomized Trees 회귀

```
model = make_pipeline(
```

```
    StandardScaler(),
    ExtraTreesRegressor()
)
```

```
cross_val = cross_validate(
    estimator=model,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.1796281337738037 (+/- 0.002060360693749536)
avg score time: 0.007660436630249024 (+/- 0.0001810064305721467)
avg test score: 0.635732588813579 (+/- 0.24707494283874556)
```

```
cross_val = cross_validate(
    estimator=model,
    X=diabetes.data, y=diabetes.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.1559168815612793 (+/- 0.002655457394132586)
avg score time: 0.008100175857543945 (+/- 0.0006419414393635727)
avg test score: 0.4350362956842 (+/- 0.032315648350392946)
```

Random Forest, Extra Tree 시각화

- 결정 트리, Random Forest, Extra Tree의 결정 경계와 회귀식 시각화

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use(['seaborn-whitegrid'])
from matplotlib.colors import ListedColormap
from sklearn.tree import DecisionTreeClassifier
```

```
n_classes = 3
n_estimators = 30
cmap = plt.cm.RdYlBu
plot_step = 0.02
plot_step_coarser = 0.5
RANDOM_SEED = 13
```

```
iris = load_iris()
plot_idx = 1
models = [DecisionTreeClassifier(max_depth=None),
          RandomForestClassifier(n_estimators=n_estimators),
```

```
          ExtraTreesClassifier(n_estimators=n_estimators)]
```

```
plt.figure(figsize=(16, 8))

for pair in ([0, 1], [0, 2], [2, 3]):

  for model in models:

    X = iris.data[:, pair]
    y = iris.target

    idx = np.arange(X.shape[0])
    np.random.seed(RANDOM_SEED)
    np.random.shuffle(idx)
    X = X[idx]
    y = y[idx]

    mean = X.mean(axis=0)
    std = X.std(axis=0)
    X = (X - mean) / std

    model.fit(X, y)

    model_title = str(type(model)).split(".")[-1][:-2][:-len("Classifier")]

    plt.subplot(3, 3, plot_idx)
    if plot_idx <= len(models):
      plt.title(model_title, fontsize=9)

    x_min, x_max = X[:, 0].min()-1, X[:, 0].max()+1
    y_min, y_max = X[:, 1].min()-1, X[:, 1].max()+1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))

    if isinstance(model, DecisionTreeClassifier):
      Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
      Z = Z.reshape(xx.shape)
      cs = plt.contourf(xx, yy, Z, cmap=cmap)
    else:
      estimator_alpha = 1.0 / len(model.estimators_)
      for tree in model.estimators_:
        Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        cs = plt.contourf(xx, yy, Z, alpha=estimator_alpha, cmap=cmap)

    xx_coarser, yy_coarser = np.meshgrid(np.arange(x_min, x_max, plot_step_coarser),
                                         np.arange(y_min, y_max, plot_step_coarser))
    Z_points_coarser = model.predict(np.c_[xx_coarser.ravel(),
                                           yy_coarser.ravel()]).reshape(xx_coarser.shape)
    cs_points = plt.scatter(xx_coarser, yy_coarser, s=15,
                            c=Z_points_coarser, cmap=cmap,
                            edgecolor='none')

    plt.scatter(X[:, 0], X[:, 1], c=y,
```

```
                cmap=ListedColormap(['r', 'y', 'b']),
                edgecolor='k', s=20)
    plot_idx += 1

plt.suptitle("Classifiers", fontsize=12)
plt.axis('tight')
plt.tight_layout(h_pad=0.2, w_pad=0.2, pad=2.5)
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-29-71a7c4c7cbfd> in <module>()
     22     model_title = str(type(model)).split(".")[-1][:-2][:-len("Classifier")]
     23
---> 24     plt.subplot(3, 7, plot_idx)
     25     if plot_idx <= len(models):
     26       plt.title(model_title, fontsize=9)

 2 frames
/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_subplots.py in
__init__(self, fig, *args, **kwargs)
     64                 if num < 1 or num > rows*cols:
     65                     raise ValueError(
---> 66                         f"num must be 1 <= num <= {rows*cols}, not {num}")
     67                 self._subplotspec = GridSpec(
     68                         rows, cols, figure=self.figure)[int(num) - 1]

ValueError: num must be 1 <= num <= 21, not 37

SEARCH STACK OVERFLOW
```



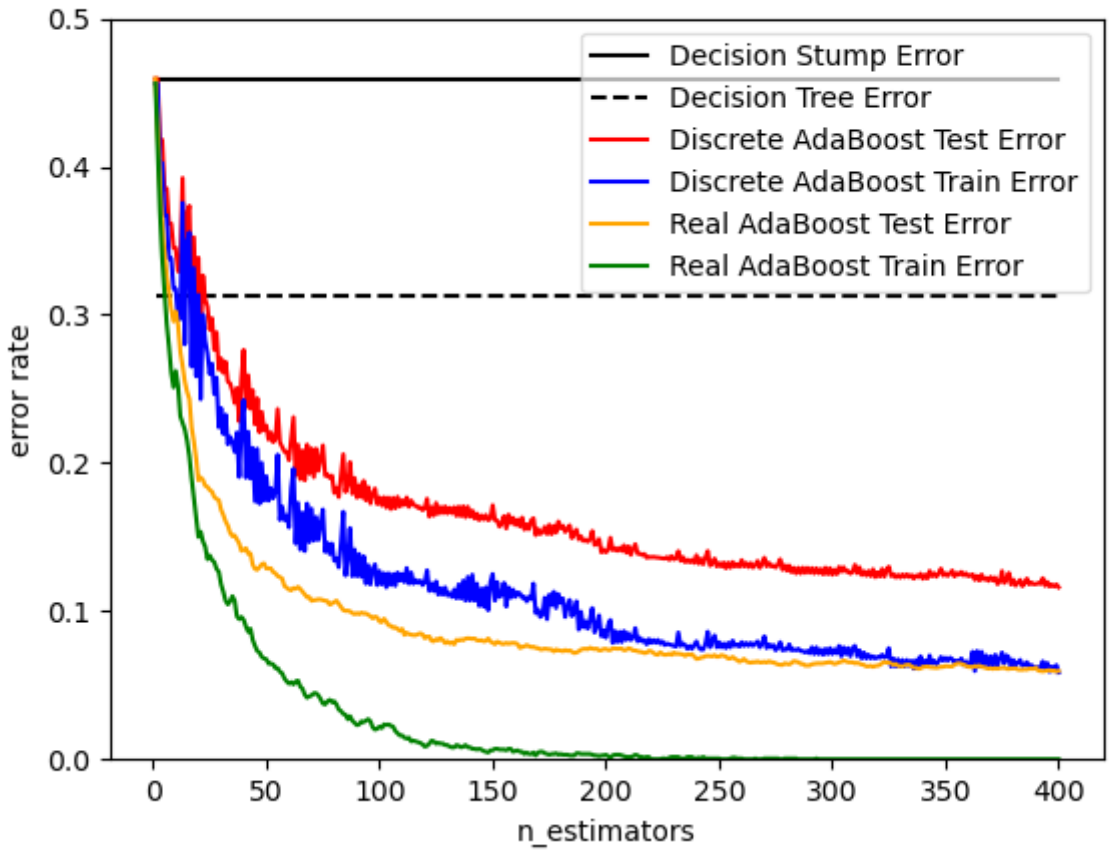
```
plot_idx = 1
models = [DecisionTreeRegressor(max_depth=None),
          RandomForestRegressor(n_estimators=n_estimators),
          ExtraTreesRegressor(n_estimators=n_estimators)]
```

```
plt.figure(figsize=(16, 8))

for pair in (0, 1, 2):

  for model in models:

    X = boston.data[:, pair]
    y = boston.target

    idx = np.arange(X.shape[0])
    np.random.seed(RANDOM_SEED)
    np.random.shuffle(idx)
    X = X[idx]
    y = y[idx]

    mean = X.mean(axis=0)
    std = X.std(axis=0)
    X = (X - mean) / std
```

```
    model.fit(X.reshape(-1, 1), y)

    model_title = str(type(model)).split(".")[-1][:-2][:-len('Regressor')]

    plt.subplot(3, 3, plot_idx)
    if plot_idx <= len(models):
      plt.title(model_title, fontsize=9)

    x_min, x_max = X.min()-1, X.max()+1
    y_min, y_max = y.min()-1, y.max()+1
    xx, yy = np.arange(x_min-1, x_max+1, plot_step), np.arange(y_min-1, y_max+1, plot_step)

    if isinstance(model, DecisionTreeRegressor):
      Z = model.predict(xx.reshape(-1, 1))
      cs = plt.plot(xx, Z)
    else:
      estimator_alpha = 1.0 / len(model.estimators_)
      for tree in model.estimators_:
        Z = tree.predict(xx.reshape(-1, 1))
        cs= plt.plot(xx, Z, alpha=estimator_alpha)

    plt.scatter(X, y, edgecolors='k', s=20)
    plot_idx += 1

plt.suptitle("Regressor", fontsize=12)
plt.axis('tight')
plt.tight_layout(h_pad=0.2, w_pad=0.2, pad=2.5)
plt.show()
```

AdaBoost

- 대표적인 부스팅 알고리즘
- 일련의 약한 모델들을 학습
- 수정된 버전의 데이터를 반복 학습 (가중치가 적용된)
- 가중치 투표(또는 합)을 통해 각 모델의 예측 값을 결합
- 첫 단계에서는 원본 데이터를 학습하고 연속적인 반복마다 개별 샘플에 대한 가중치가 수정되고 다시 모델이 학습
    - 잘못 예측된 샘플은 가중치 증가, 올바르게 예측된 샘플은 가중치 감소
    - 각각의 약한 모델들은 예측하기 어려운 샘플에 집중하게 됨




```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import AdaBoostRegressor
```

AdaBoost 분류

```
model = make_pipeline(
    StandardScaler(),
    AdaBoostClassifier()
)
```

```
cross_val = cross_validate(
    estimator=model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.06470780372619629 (+/- 0.0024589582450077756)
avg score time: 0.006348896026611328 (+/- 0.00020403328768472522)
avg test score: 0.9466666666666667 (+/- 0.03399346342395189)
```

```
cross_val = cross_validate(
    estimator=model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.07434139251708985 (+/- 0.004355340981360118)
avg score time: 0.006343555450439453 (+/- 0.00011292540898904448)
avg test score: 0.8085714285714285 (+/- 0.16822356718459935)
```

```
cross_val = cross_validate(
    estimator=model,
    X=cancer.data, y=cancer.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.15188159942626953 (+/- 0.006063555430320384)
avg score time: 0.007553672790527344 (+/- 0.0002754285311434353)
avg test score: 0.9701133364384411 (+/- 0.019709915473893072)
```

AdaBoost 회귀

```
model = make_pipeline(
    StandardScaler(),
    AdaBoostRegressor()
)
```

```
cross_val = cross_validate(
    estimator=model,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
```

```
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.09566020965576172 (+/- 0.004728853628844173)
avg score time: 0.003963184356689453 (+/- 0.00014140038108262842)
avg test score: 0.5985744583807469 (+/- 0.20937548598257683)
```

```
cross_val = cross_validate(
    estimator=model,
    X=diabetes.data, y=diabetes.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.08189129829406738 (+/- 0.016465648555033315)
avg score time: 0.003926944732666015 (+/- 0.0008019656425732309)
avg test score: 0.41312084500745616 (+/- 0.04091835944493939)
```

Gradient Tree Boosting

- 임의의 차별화 가능한 손실함수로 일반화한 부스팅 알고리즘
- 웹 검색, 분류 및 회귀 등 다양한 분야에서 모두 사용 가능

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
```

Gradient Tree Boosting 분류

```
model = make_pipeline(
    StandardScaler(),
    GradientBoostingClassifier()
)
```

```
cross_val = cross_validate(
    estimator=model,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.1825028419494629 (+/- 0.012155321171336445)
avg score time: 0.0009828567504882812 (+/- 5.27573717906438e-05)
avg test score: 0.96 (+/- 0.024944382578492935)
```

```
cross_val = cross_validate(
```

```
    estimator=model,
    X=wine.data, y=wine.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.22722344398498534 (+/- 0.006153967604831455)
avg score time: 0.000991487503051758 (+/- 4.429727374789129e-05)
avg test score: 0.9330158730158731 (+/- 0.04127777701479872)
```

```
cross_val = cross_validate(
    estimator=model,
    X=cancer.data, y=cancer.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.42472524642944337 (+/- 0.00726098838890154)
avg score time: 0.000966119766235351 5 (+/- 2.5288738491641932e-05)
avg test score: 0.9596180717279925 (+/- 0.02453263202329889)
```

Gradient Tree Boosting 회귀

```
model = make_pipeline(
    StandardScaler(),
    GradientBoostingRegressor()
)
```

```
cross_val = cross_validate(
    estimator=model,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.11993212699890136 (+/- 0.0025604519552051906)
avg score time: 0.000992870330810546 8 (+/- 3.9700409139679184e-05)
avg test score: 0.6752372690883461 (+/- 0.16121836382662494)
```

```
cross_val = cross_validate(
    estimator=model,
    X=diabetes.data, y=diabetes.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
```

```
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.0954463005065918 (+/- 0.005929260406269451)
avg score time: 0.0009887218475341797 (+/- 2.1540693105770847e-05)
avg test score: 0.4029450867301339 (+/- 0.0688801507855847)
```

투표 기반 분류 (Voting Classifier)

- 서로 다른 모델들의 결과를 투표를 통해 결합
- 두가지 방법으로 투표 가능
  - 가장 많이 예측된 클래스를 정답으로 채택 (hard voting)
  - 예측된 확률의 가중치 평균 (soft voting)

```
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import cross_val_score
```

```
model1 = SVC()
model2 = GaussianNB()
model3 = RandomForestClassifier()
vote_model = VotingClassifier(
    estimators=[('svc', model1), ('naive', model2), ('forest', model3)],
    voting='hard'
)
```

```
for model in (model1, model2, model3, vote_model):
  model_name = str(type(model)).split('.')[-1][:-2]
  scores = cross_val_score(model, iris.data, iris.target, cv=5)
  print('Accuracy: %0.2f (+/- %0.2f) [%s]' % (scores.mean(), scores.std(), model_name))
```

```
Accuracy: 0.97 (+/- 0.02) [SVC]
Accuracy: 0.95 (+/- 0.03) [GaussianNB]
Accuracy: 0.97 (+/- 0.02) [RandomForestClassifier]
Accuracy: 0.97 (+/- 0.02) [VotingClassifier]
```

```
model1 = SVC(probability=True)
model2 = GaussianNB()
model3 = RandomForestClassifier()
vote_model = VotingClassifier(
    estimators=[('svc', model1), ('naive', model2), ('forest', model3)],
    voting='soft',
    weights=[2, 1, 2]
)
```

```
for model in (model1, model2, model3, vote_model):
  model_name = str(type(model)).split('.')[-1][:-2]
  scores = cross_val_score(model, iris.data, iris.target, cv=5)
  print('Accuracy: %0.2f (+/- %0.2f) [%s]' % (scores.mean(), scores.std(), model_name))
```

```
  print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), model_name))
```

```
Accuracy: 0.97 (+/- 0.02) [SVC]
Accuracy: 0.95 (+/- 0.03) [GaussianNB]
Accuracy: 0.95 (+/- 0.03) [RandomForestClassifier]
Accuracy: 0.96 (+/- 0.02) [VotingClassifier]
```

결정 경계 시각화

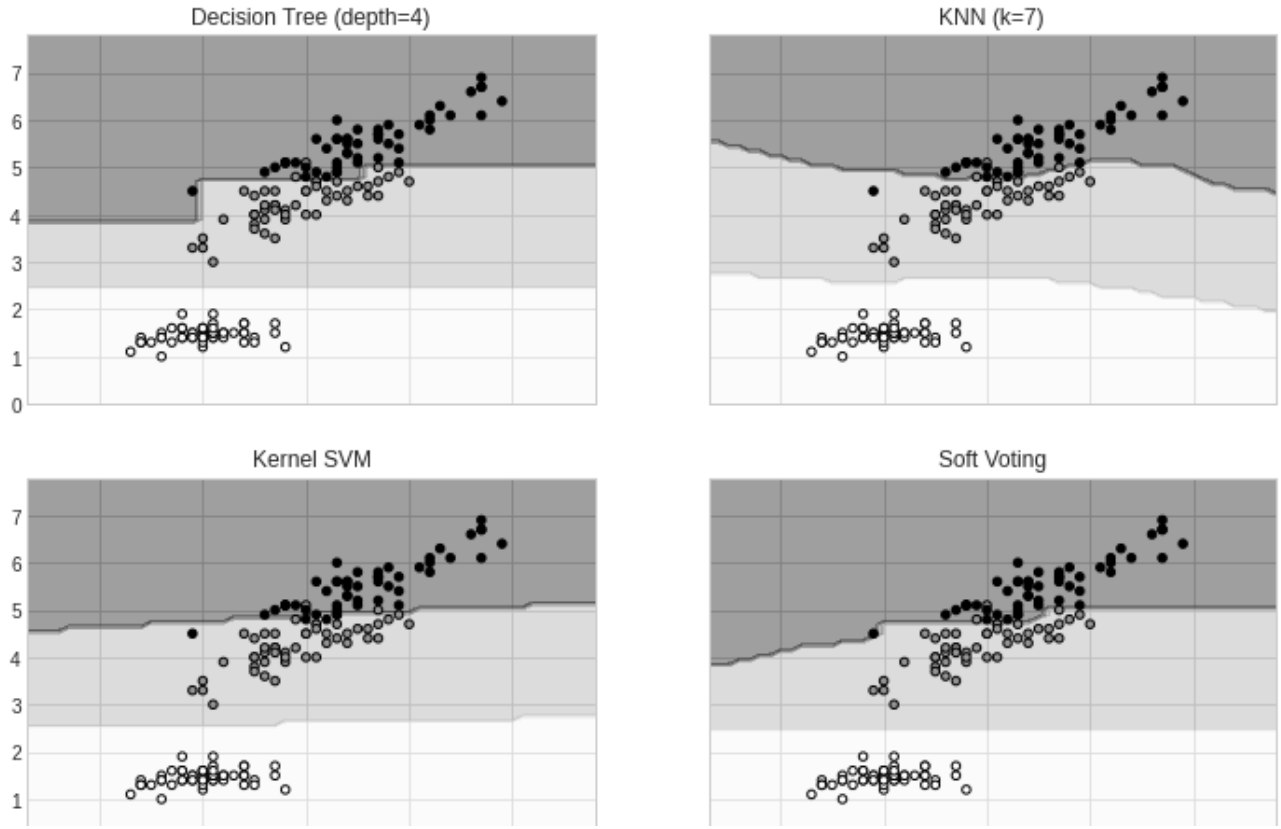
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from itertools import product
```

```
X = iris.data[:, [0, 2]]
y = iris.target
```

```
model1 = DecisionTreeClassifier(max_depth=4)
model2 = KNeighborsClassifier(n_neighbors=7)
model3 = SVC(gamma=.1, kernel='rbf', probability=True)
vote_model = VotingClassifier(estimators=[('dt', model1), ('knn', model2), ('svc', model3)],
                              voting='soft', weights=[2, 1, 2])
model1 = model1.fit(X, y)
model2 = model2.fit(X, y)
model3 = model3.fit(X, y)
vote_model = vote_model.fit(X, y)
```

```
x_min, x_max = X[:, 0].min()-1, X[:, 0].max()+1
y_min, y_max = X[:, 1].min()-1, X[:, 1].max()+1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
```

```
f, axarr = plt.subplots(2, 2, sharex='col', sharey='row', figsize=(12, 8))

for idx, model, tt in zip(product([0, 1], [0, 1]),
                          [model1, model2, model3, vote_model],
                          ['Decision Tree (depth=4)', 'KNN (k=7)',
                           'Kernel SVM', 'Soft Voting']):
  Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
  Z = Z.reshape(xx.shape)

  axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.4)
  axarr[idx[0], idx[1]].scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
  axarr[idx[0], idx[1]].set_title(tt)

plt.show()
```



투표 기반 회귀 (Voting Regressor)

- 서로 다른 모델의 예측 값의 평균을 사용

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import VotingRegressor
```

```
model1 = LinearRegression()
model2 = GradientBoostingRegressor()
model3 = RandomForestRegressor()
vote_model = VotingRegressor(
    estimators=[('linear', model1), ('gbr', model2), ('rfr', model3)],
    weights=[1, 1, 1]
)
```

```
for model in (model1, model2, model3, vote_model):
  model_name = str(type(model)).split('.')[-1][:-2]
  scores = cross_val_score(model, boston.data, boston.target, cv=5)
  print('R2: %0.2f (+/- %0.2f) [%s]' % (scores.mean(), scores.std(), model_name))
```

```
R2: 0.35 (+/- 0.38) [LinearRegression]
R2: 0.67 (+/- 0.16) [GradientBoostingRegressor]
R2: 0.63 (+/- 0.21) [RandomForestRegressor]
R2: 0.66 (+/- 0.20) [VotingRegressor]
```

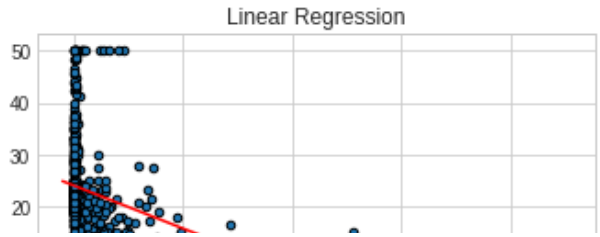
회귀식 시각화

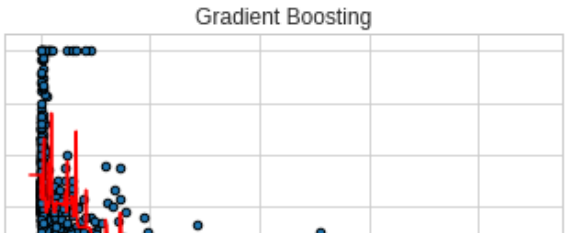
```
X = boston.data[:, 0].reshape(-1, 1)
y = boston.target
```

```
model1 = LinearRegression()
model2 = GradientBoostingRegressor()
model3 = RandomForestRegressor()
vote_model = VotingRegressor(
    estimators=[('linear', model1), ('gbr', model2), ('rfr', model3)],
    weights=[1, 1, 1]
)

model1 = model1.fit(X, y)
model2 = model2.fit(X, y)
model3 = model3.fit(X, y)
vote_model = vote_model.fit(X, y)
```

```
x_min, x_max = X.min()-1, X.max()+1
xx = np.arange(x_min-1, x_max+1, 0.1)
```

```
f, axarr = plt.subplots(2, 2, sharex='col', sharey='row', figsize=(12, 8))

for idx, model, tt in zip(product([0, 1], [0, 1]),
                          [model1, model2, model3, vote_model],
                          ['Linear Regression', 'Gradient Boosting', 'Random Forest', 'Voting']):
  Z = model.predict(xx.reshape(-1, 1))

  axarr[idx[0], idx[1]].plot(xx, Z, c='r')
  axarr[idx[0], idx[1]].scatter(X, y, s=20, edgecolor='k')
  axarr[idx[0], idx[1]].set_title(tt)

plt.show()
```





스택 일반화 (Stacked Generalization)

- 각 모델의 예측 값을 최종 모델의 입력으로 사용
- 모델의 편향을 줄이는데 효과적

스택 회귀

```
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import StackingRegressor
```


```
estimators = [('ridge', Ridge()),
              ('lasso', Lasso()),
              ('svr', SVR())]
```

```
reg = make_pipeline(
    StandardScaler(),
    StackingRegressor(
        estimators=estimators,
        final_estimator=GradientBoostingRegressor()
    )
)
```

```
cross_val = cross_validate(
    estimator=reg,
    X=boston.data, y=boston.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std())
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

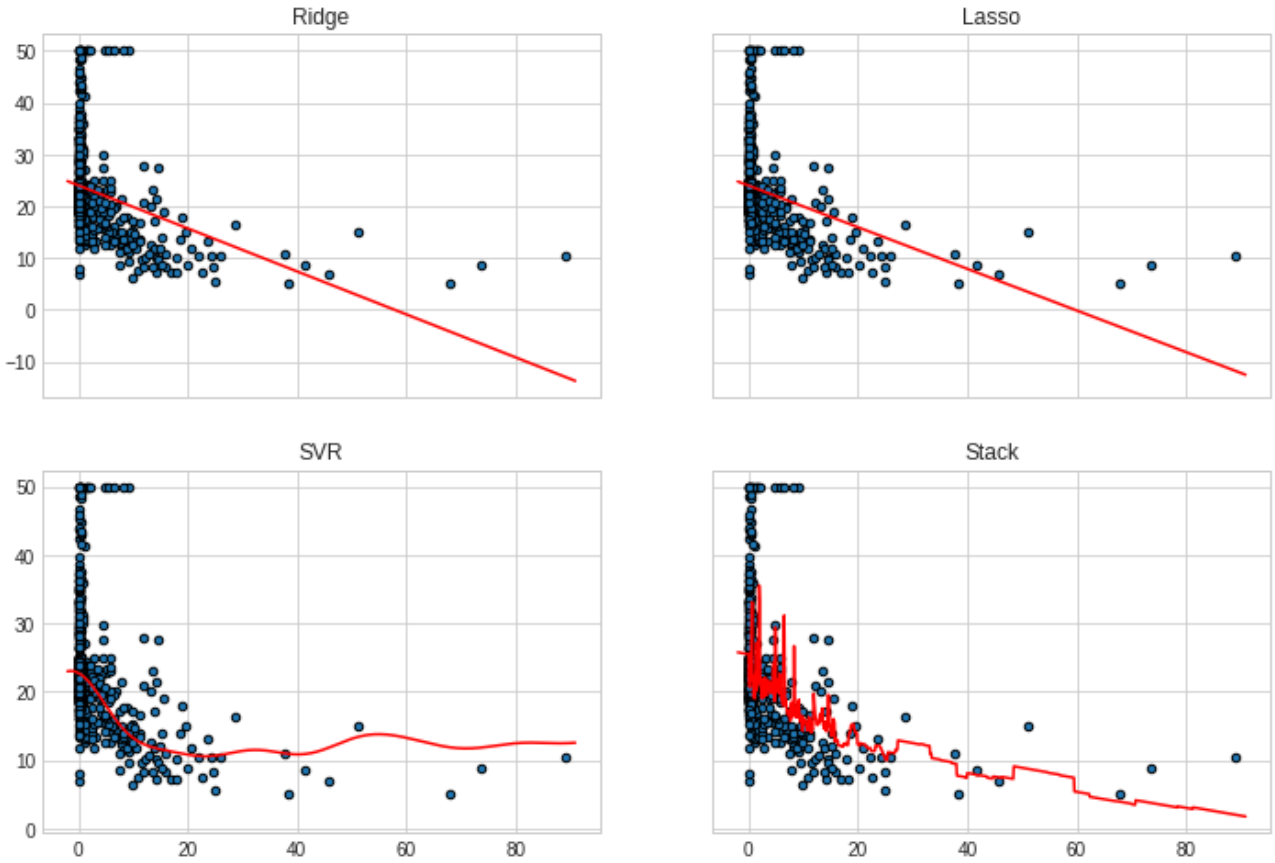
```
avg fit time: 0.14769234657287597 (+/- 0.003973230032571223)
avg score time: 0.0029734134674072265 (+/- 8.165947363439729e-05)
avg test score: 0.3193125598078287 (+/- 0.3328757426063098)
```

회귀식 시각화

```
X = boston.data[:, 0].reshape(-1, 1)
y = boston.target
```

```
model1 = Ridge()
model2 = Lasso()
model3 = SVR()
reg = StackingRegressor(
        estimators=estimators,
        final_estimator=GradientBoostingRegressor()
)

model1 = model1.fit(X, y)
model2 = model2.fit(X, y)
model3 = model3.fit(X, y)
reg = reg.fit(X, y)
```

```
f, axarr = plt.subplots(2, 2, sharex='col', sharey='row', figsize=(12, 8))

for idx, model, tt in zip(product([0, 1], [0, 1]),
                          [model1, model2, model3, vote_model],
                          ['Ridge', 'Lasso', 'SVR', 'Stack']):
  Z = model.predict(xx.reshape(-1, 1))

  axarr[idx[0], idx[1]].plot(xx, Z, c='r')
  axarr[idx[0], idx[1]].scatter(X, y, s=20, edgecolor='k')
  axarr[idx[0], idx[1]].set_title(tt)

plt.show()
```



스택 분류

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier
```

```
estimators = [('logistic', LogisticRegression(max_iter=10000)),
              ('svc', SVC()),
              ('naive', GaussianNB())]
```

```
clf = StackingClassifier(
    estimators=estimators,
    final_estimator=RandomForestClassifier()
)
```

```
cross_val = cross_validate(
    estimator=clf,
    X=iris.data, y=iris.target,
    cv=5
)
print('avg fit time: {} (+/- {})'.format(cross_val['fit_time'].mean(), cross_val['fit_time'].std()
print('avg score time: {} (+/- {})'.format(cross_val['score_time'].mean(), cross_val['score_time'].
print('avg test score: {} (+/- {})'.format(cross_val['test_score'].mean(), cross_val['test_score'].
```

```
avg fit time: 0.2862356185913086 (+/- 0.007009096495986199)
avg score time: 0.008571338653564454 (+/- 0.00010991095462228061)
avg test score: 0.96 (+/- 0.024944382578492935)
```

결정 경계 시각화

```
X = iris.data[:, [0, 2]]
y = iris.target
```

```
model1 = LogisticRegression(max_iter=10000)
model2 = SVC()
model3 = GaussianNB()
stack = StackingClassifier(
    estimators=estimators,
    final_estimator=RandomForestClassifier()
)

model1 = model1.fit(X, y)
model2 = model2.fit(X, y)
model3 = model3.fit(X, y)
stack = stack.fit(X, y)
```

```
x_min, x_max = X[:, 0].min()-1, X[:, 0].max()+1
y_min, y_max = X[:, 1].min()-1, X[:, 1].max()+1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
```

```
f, axarr = plt.subplots(2, 2, sharex='col', sharey='row', figsize=(12, 8))

for idx, model, tt in zip(product([0, 1], [0, 1]),
                          [model1, model2, model3, stack],
                          ['Logistic Regression', 'SVC',
                           'GaussianNB', 'Stack']):
  Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
  Z = Z.reshape(xx.shape)

  axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.4)
  axarr[idx[0], idx[1]].scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
  axarr[idx[0], idx[1]].set_title(tt)

plt.show()
```