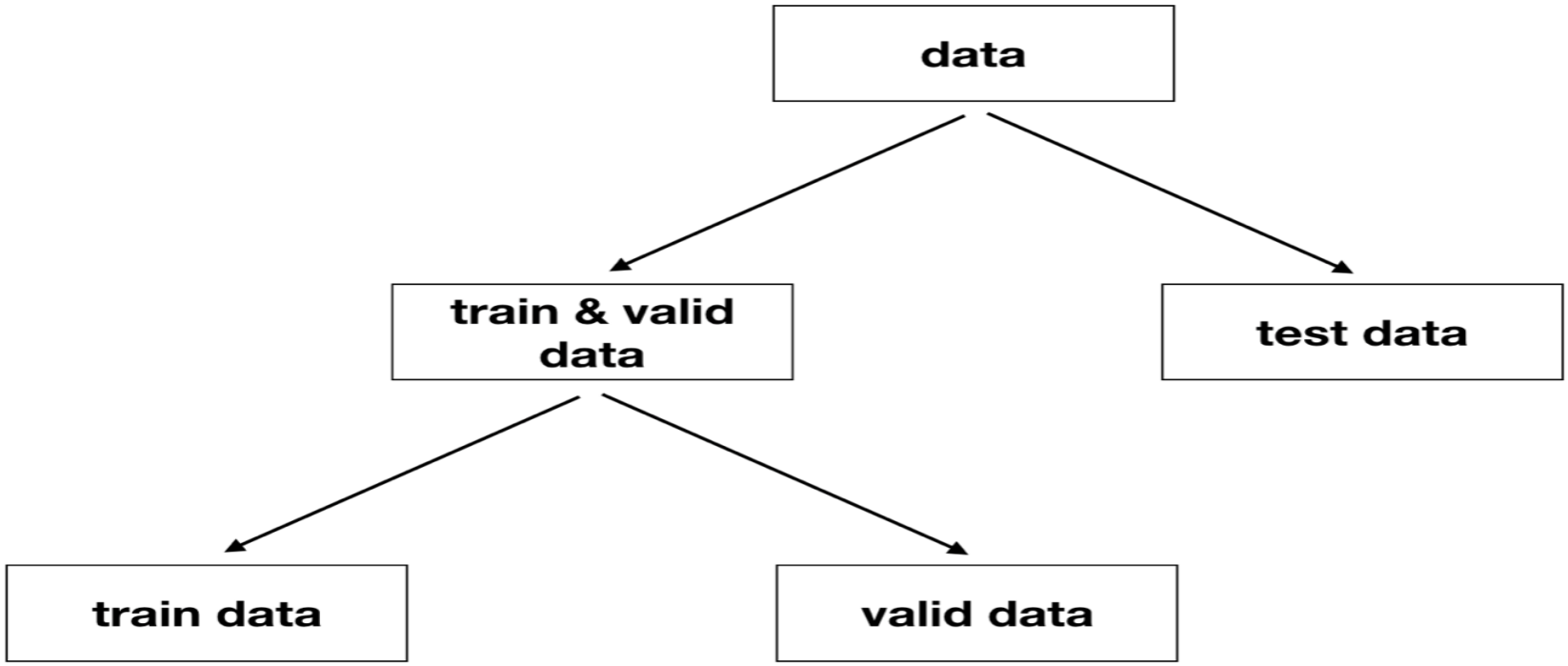
모델 검증과 앙상블

교차 검증

교차검증

□ 훈련 검증 테스트셋 분리

- 훈련데이터: 모델을 학습하는데 사용하는 데이터
- 검증데이터: 학습한 모델의 성능을 검증하는 데이터 (모델이 모르는 데이터로 하이퍼 파라미터의 선택에 사용된다.)
- 테스트데이터: 선택된 하이퍼 파라미터로 학습된 모델로 예측할 때의 데이터로 모델이 모르는 데이터

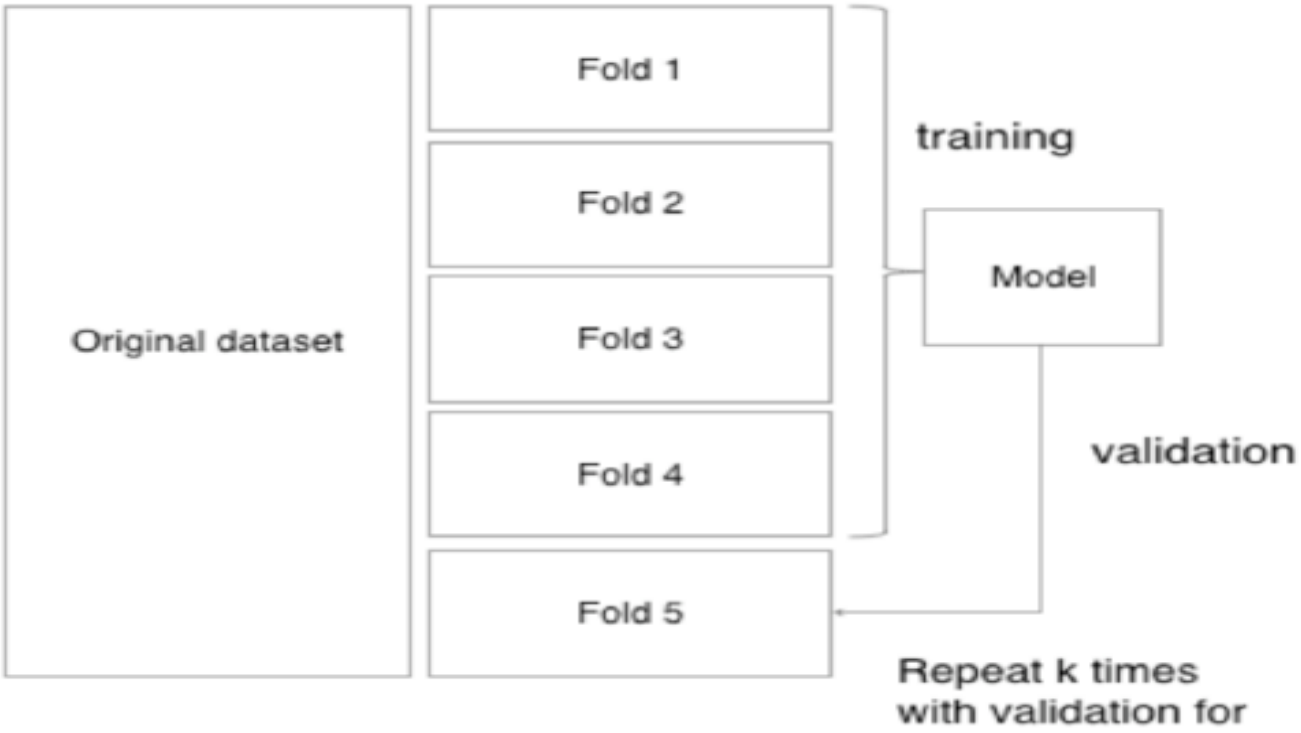


교차검증

□ K-fold with stratify (층화 k겹)

k-fold는 데이터를 k개로 분할하는 것으로 교차검증에 사용되는데, 데이터를 k개로 분할해 k-1개로 모델을 학습하고 나머지 1개로 모델을 검증한다.

k개로 분할하면, 모든 겹(fold)에 대해서 하나의 겹을 선택하는 방식으로 k번 다른 데이터셋으로 학습한 모델을 검증할 수 있다.

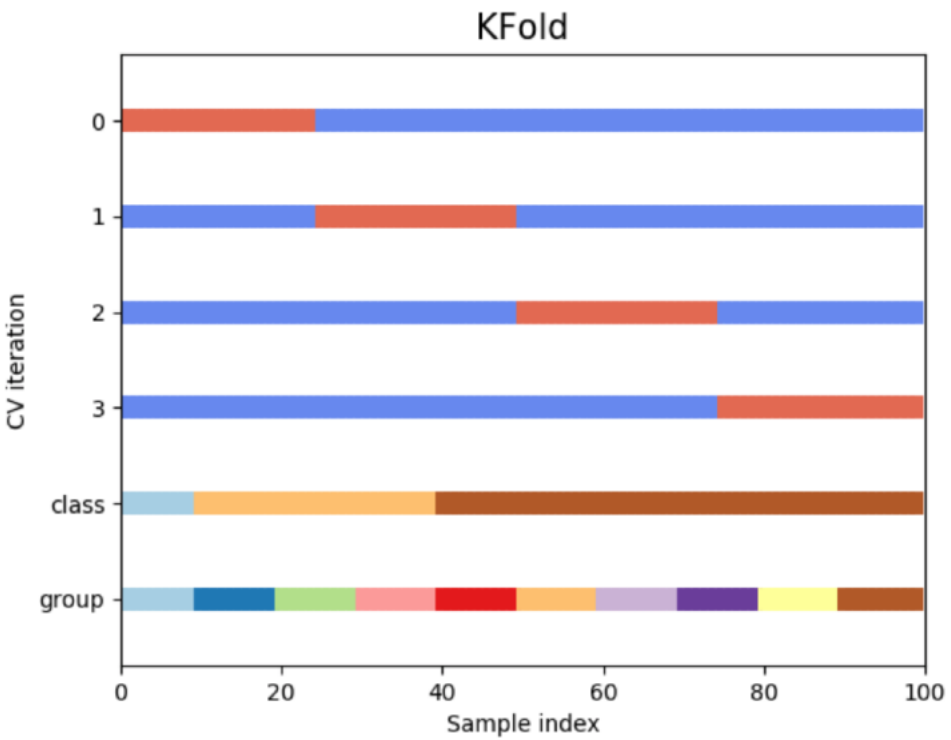


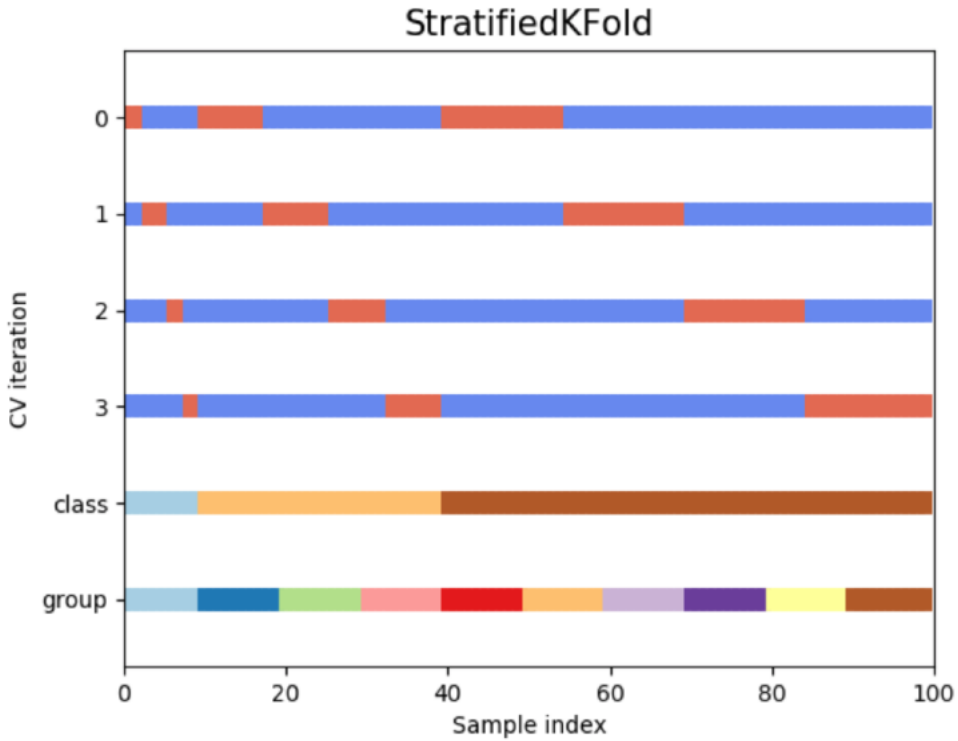
교차검증

□ stratify의 의미

k겹은 데이터의 정렬 유무와 분류할 클래스의 비율에 상관없이 데이터를 분할한다. 분류할 클래스의 비율이 다르면 각 fold가 학습 데이터셋을 대표한다고 하기 어렵다. 한 fold에 특정 클래스가 많이 나올수도 적게 나올수도 있기 때문이다.

stratified k-fold는 이런 문제를 해결하기위해 제안된 것으로 k개의 fold로 분할한 이후에도 전체 훈련데이터셋의 클래스비율과 각 fold의 클래스 비율을 동일하게 맞춘다는 점이 기존 k-fold와 다른 점이다.





교차검증

□ KFold

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5, random_state=2019)
```

```
for i, (trn_idx, val_idx) in enumerate(kf.split(kf_data.values, kf_label)) :
    trn_data, trn_label = kf_data.values[trn_idx,:], kf_label[trn_idx]
    val_data, val_label = kf_data.values[val_idx,:], kf_label[val_idx]

    print('{} Fold, trn label\n {}'.format(i, trn_label))
    print('{} Fold, val label\n {}\n'.format(i, val_label))
```

```
0 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
0 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

1 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
1 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
 1 1 1 1 1 1 1 1 1]

2 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
2 Fold, val label
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
3 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
3 Fold, val label
 [1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

4 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
4 Fold, val label
 [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

교차검증

□ StratifiedKFold

```python
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, random_state=2019)
```

```python
for i, (trn_idx, val_idx) in enumerate(skf.split(kf_data.values, kf_label)) :
    trn_data, trn_label = kf_data.values[trn_idx,:], kf_label[trn_idx]
    val_data, val_label = kf_data.values[val_idx,:], kf_label[val_idx]

    print('{} Fold, trn label\n {}'.format(i, trn_label))
    print('{} Fold, val label\n {}\n'.format(i, val_label))
```

```
0 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
0 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]

1 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
1 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]

2 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
2 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]

3 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
3 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]

4 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
4 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]
```

교차검증

□ 교차검증 해보기

```
from sklearn.ensemble import RandomForestClassifier

val_scores = list()

for i, (trn_idx, val_idx) in enumerate(skf.split(kf_data, kf_label)) :
    trn_data, trn_label = kf_data.values[trn_idx,:], kf_label[trn_idx]
    val_data, val_label = kf_data.values[val_idx,:], kf_label[val_idx]

    # 모델 정의
    clf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=2019)

    # 모델 학습
    clf.fit(trn_data, trn_label)
    trn_acc = clf.score(trn_data, trn_label)*100
    val_acc = clf.score(val_data, val_label)*100
    print('{} Fold, train Accuracy : {:.2f}%, validation Accuracy : {:.2f}%'.format(i, trn_acc, val_acc))

    val_scores.append(val_acc)

# Mean Validation Score
print('Cross Validation Score : {:.2f}%'.format(np.mean(val_scores)))
```

```
0 Fold, train Accuracy : 100.00%, validation Accuracy : 96.67%
1 Fold, train Accuracy : 100.00%, validation Accuracy : 96.67%
2 Fold, train Accuracy : 100.00%, validation Accuracy : 93.33%
3 Fold, train Accuracy : 100.00%, validation Accuracy : 96.67%
4 Fold, train Accuracy : 100.00%, validation Accuracy : 100.00%
Cross Validation Score : 96.67%
```

교차검증

□ 교차검증 점수: cross_val_score는 한번에 k-fold cross val. score를 계산한다.

-파라미터로 cv에 숫자를 전달하면, 그 숫자만큼의 fold를 만들어 CV를 진행하고 kfold 객체를 전달하면 해당 객체에 맞게 데이터를 부할해 CV 점수를 계산한다.

-cross_val_score함수는 fold 개수대로 score를 반환하며 해당 score의 평균을 계산해 모델의 성능을 평가한다.

- cross_cal_score함수는 분류모델의 경우 디폴트로 StratifiedKFold를 적용한다.

```
from sklearn.model_selection import cross_val_score
```

```
# 숫자로 전달하는 경우
print('랜덤 포레스트 k-Fold CV Score(Acc) : {:.2f}%'.format(np.mean(cross_val_score(rf, kf_data, kf_label, cv=5))*100))
```

```
랜덤 포레스트 k-Fold CV Score(Acc) : 96.00%
```

```
# fold 객체를 전달하는 경우
print('랜덤 포레스트 k-Fold CV Score(Acc) : {:.2f}%'.format(np.mean(cross_val_score(rf, kf_data, kf_label, cv=kf))*100))
print('랜덤 포레스트 Stratify k-Fold CV Score(Acc) : {:.2f}%'.format(np.mean(cross_val_score(rf, kf_data, kf_label, cv=skf))*100))
```

```
랜덤 포레스트 k-Fold CV Score(Acc) : 75.33%
랜덤 포레스트 Stratify k-Fold CV Score(Acc) : 96.00%
```

파라미터 튜닝

그리드 서치 (그리드 탐색)

□ 그리드 서치

- 모델의 하이퍼파라미터를 결정

예: SVC의 경우 소프트, 하드 마진의 정도를 결정하는 C, 커널함수를 결정하는 'kernel', 특정 커널에서 얼마나 세세하게 볼 것인가를 결정하는 'gamma'를 어떻게 결정하는가에 따라 모델의 학습 성과가 좌우될 수 있다.

- 여기서는 GridSearchCV 함수를 사용해 랜덤포레스트의 n_estimator, max_depth 파라미터 중 가장 좋은 파라미터를 찾아보는 연습을 한다.
- GridSearchCV 함수는 sklearn의 model_selection 패키지에 있다.

□ 참고

- sklearn에는 없지만, scikit-optimize(skopt)라는 라이브러리가 있다. skopt는 각 파라미터에 들어갈 값들의 최대, 최소 범위를 결정해주고, 파라미터 값의 분포 스케일을 결정해 줘 파라미터 튜닝을 자동화한다.
- skopt의 BayesSearchCV를 사용해 베이지안 하이퍼 파라미터 최적화를 할 수 있다.

그리드 서치 (그리드 탐색)

□ 그리드 서치

```
from sklearn.model_selection import GridSearchCV

params = {'n_estimators' : [50, 150, 200, 300, 400],
          'max_depth' : [2, 5 ,10, 20]}

clf = GridSearchCV(RandomForestClassifier(random_state=2019), params, cv=skf)
```

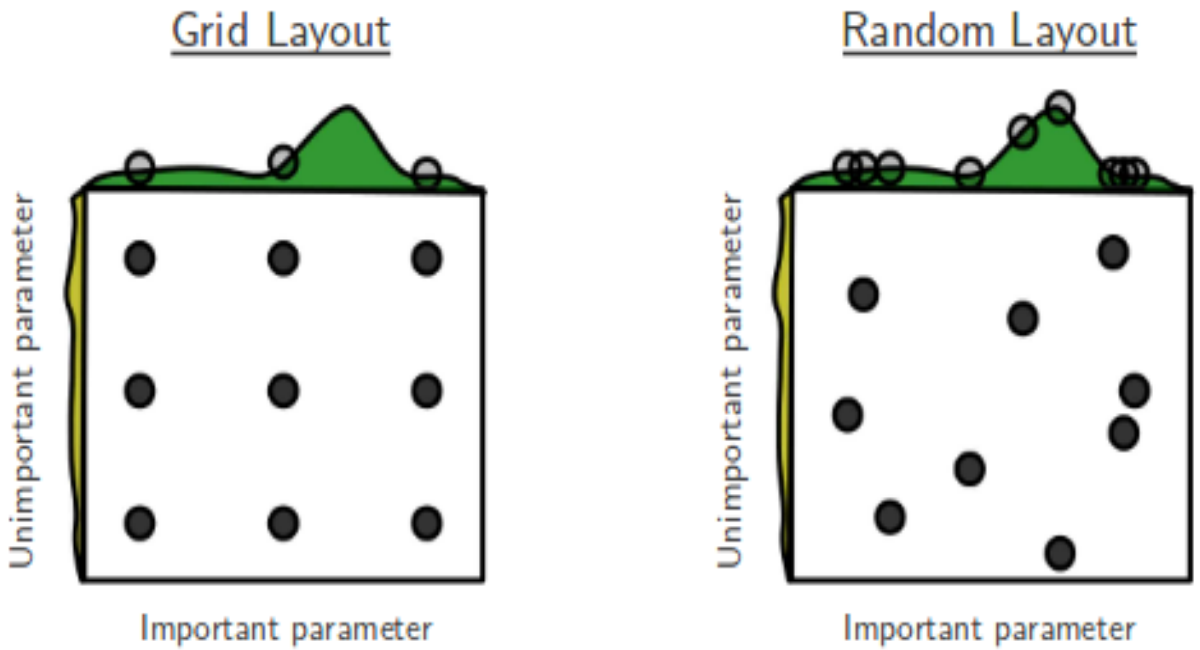
```
clf.fit(kf_data, kf_label)
```

```
print('GridSearchCV best score : {:.2f}%, best_params : {}'.format(clf.best_score_*100, clf.best_params_))
```

```
GridSearchCV best score : 96.67%, best_params : {'max_depth': 5, 'n_estimators': 50}
```

랜덤 서치 (랜덤 탐색)

□ 그리드 서치 대 랜덤 서치

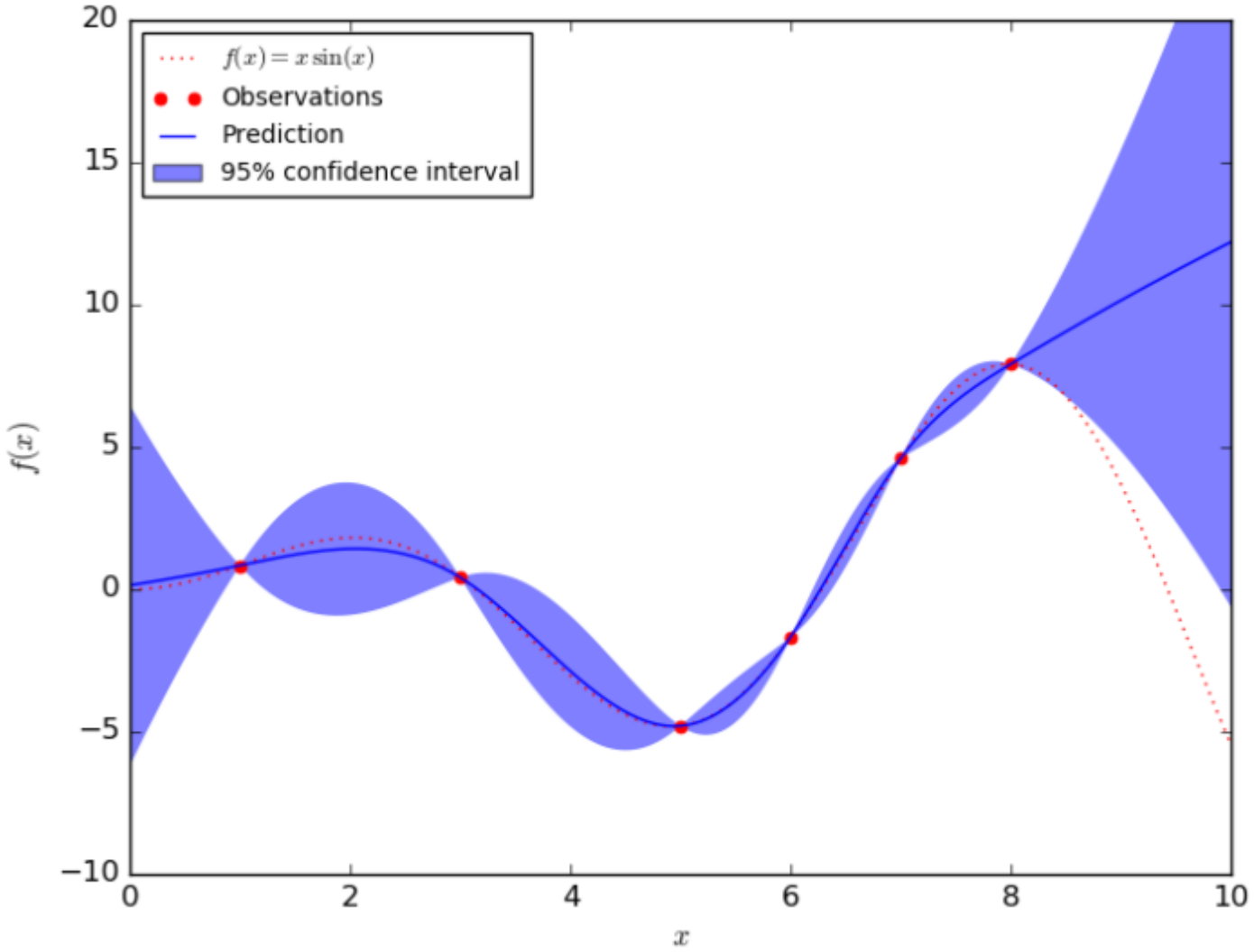


□ 랜덤 탐색 (랜덤 서치)

- sklearn.model_selecton에 RandomizedSearchCV가 있으며, 사용법은 GridSearchCV와 동일하다.

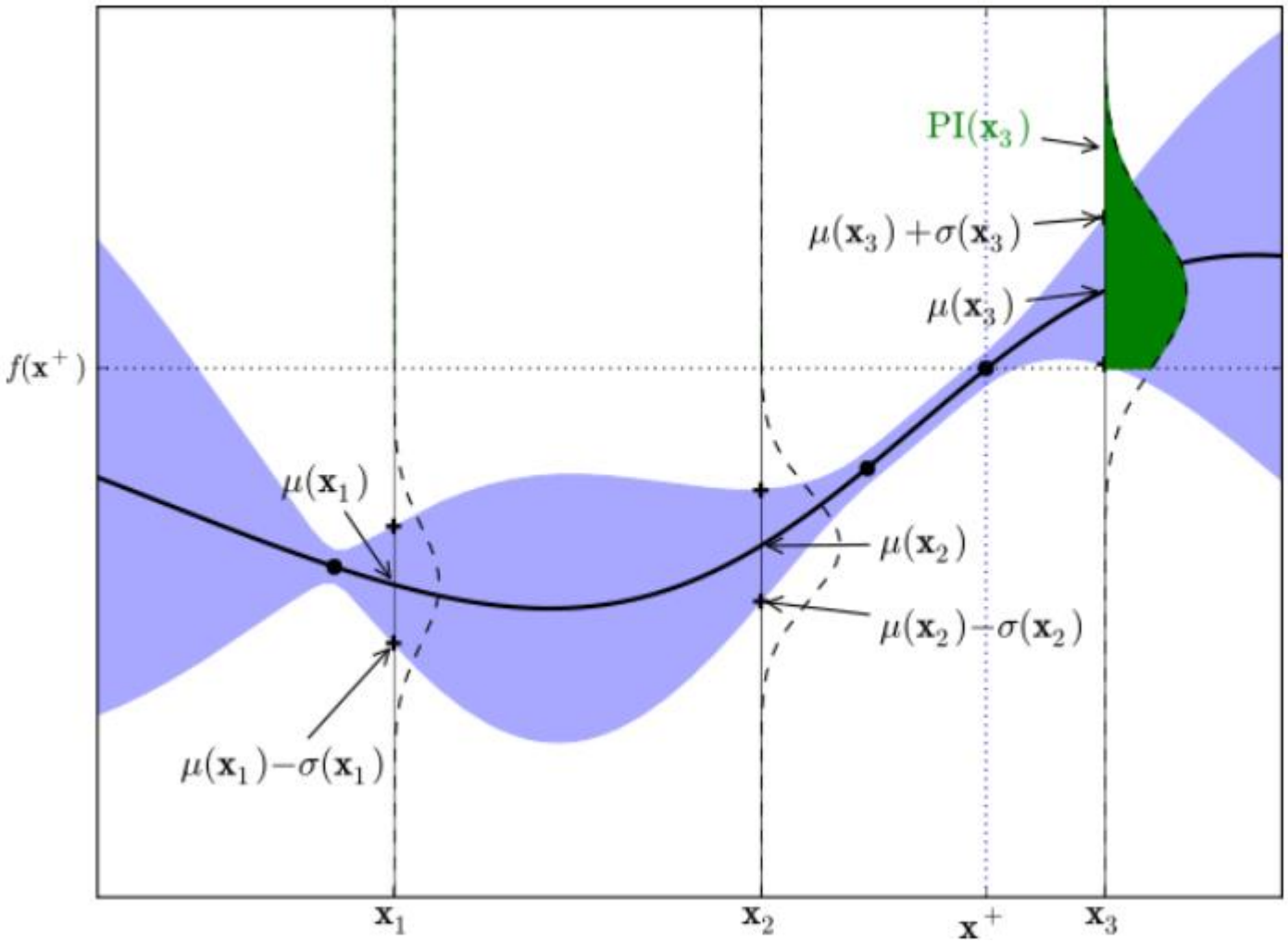
베이지안 하이퍼파라미터 최적화

□ 가우시안 프로세스 모델 - 대리모델



베이지안 하이퍼파라미터 최적화

□ 가우시안 프로세스 설명



베이지안 하이퍼파라미터 최적화

□ 획득함수(Acquisition Function)= 기대개선을 최대로 하는 함수

$$x^* = \underset{x^{new_i} \in X^{new}}{\operatorname{argmax}} \ \text{Acquisition function}(x^{new_i})$$

$$x^* = \underset{x^{new_i} \in X^{new}}{\operatorname{argmax}} \ \text{Expected Improvement}(x^{new_i})$$

$$:= E\left(\max\left(0, \left[f(x^{new_i}) - \max_{x_j \in X} f(x_j)\right]\right)\right)$$

출처 : http://dmqm.korea.ac.kr/activity/seminar/285

베이지안 하이퍼파라미터 최적화

□ 기대개선함수 = 활용(Exploitation) + 탐험(Exploration)

$$EI(x) = \mathbb{E}[\max(f(x) - f(x^+), 0)]$$

$$= \begin{cases} (\mu(\boldsymbol{x}) - f(\boldsymbol{x}^+) - \xi)\Phi(Z) + \sigma(\boldsymbol{x})\phi(Z) & \text{if } \sigma(\boldsymbol{x}) > 0 \\ 0 & \text{if } \sigma(\boldsymbol{x}) = 0 \end{cases}$$

$$Z = \begin{cases} \frac{\mu(\boldsymbol{x}) - f(\boldsymbol{x}^+) - \xi}{\sigma(\boldsymbol{x})} & \text{if } \sigma(\boldsymbol{x}) > 0 \\ 0 & \text{if } \sigma(\boldsymbol{x}) = 0 \end{cases}$$

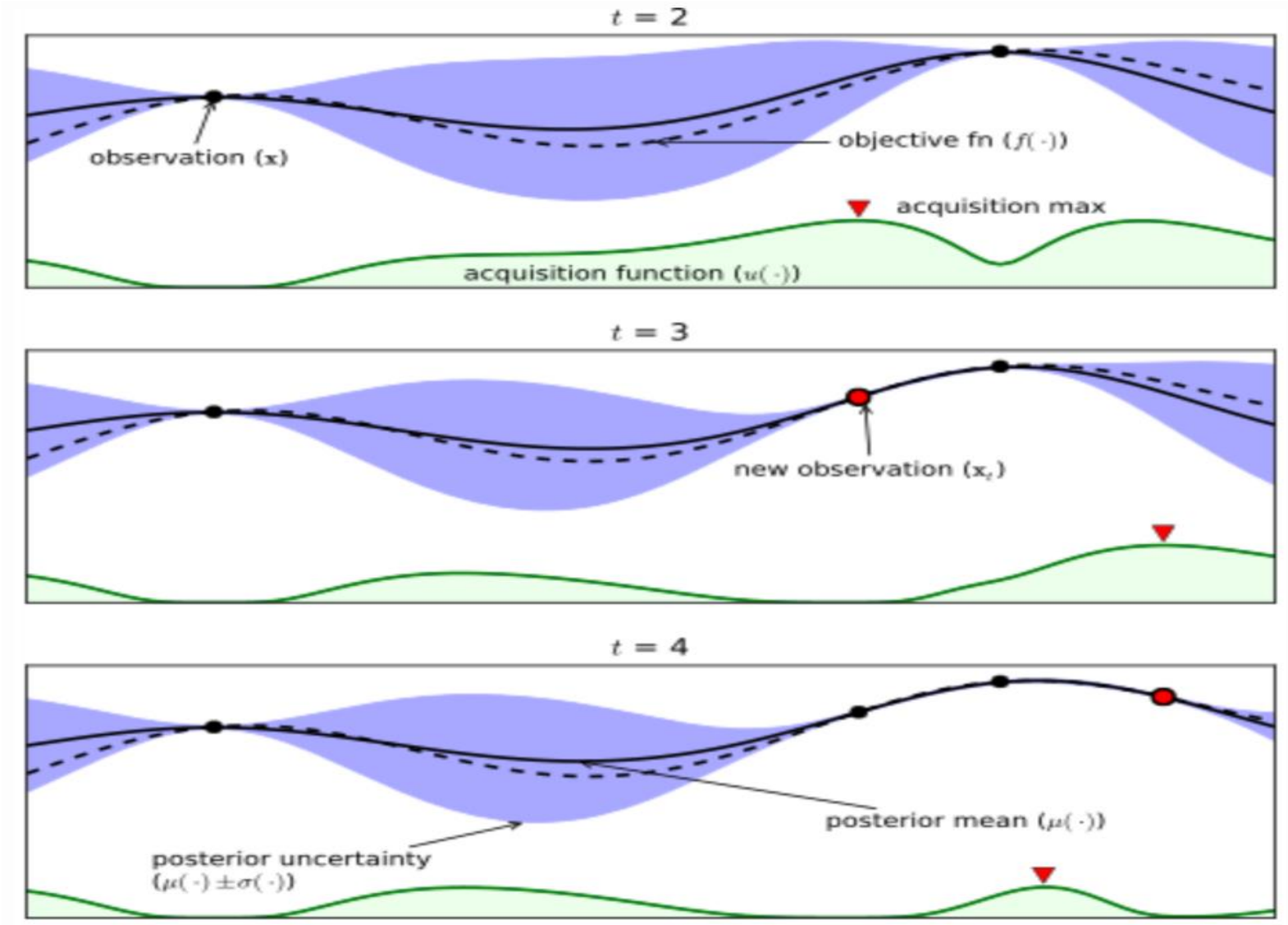
활용

탐험

베이지안 하이퍼파라미터 최적화

검은색 점선 : 실제 목적 함수
검은색 실선 : 추정된 함수의 평균
보라색 영역 : 추정된 함수의 표준편차
초록색 영역 : Acquistion Function

□ 획득함수를 활용한 대리모델의 최적화



파라미터 튜닝

□ 베이지안 하이퍼파라미터 튜닝 예

Bayesian Optimization

Feat. Cross validation

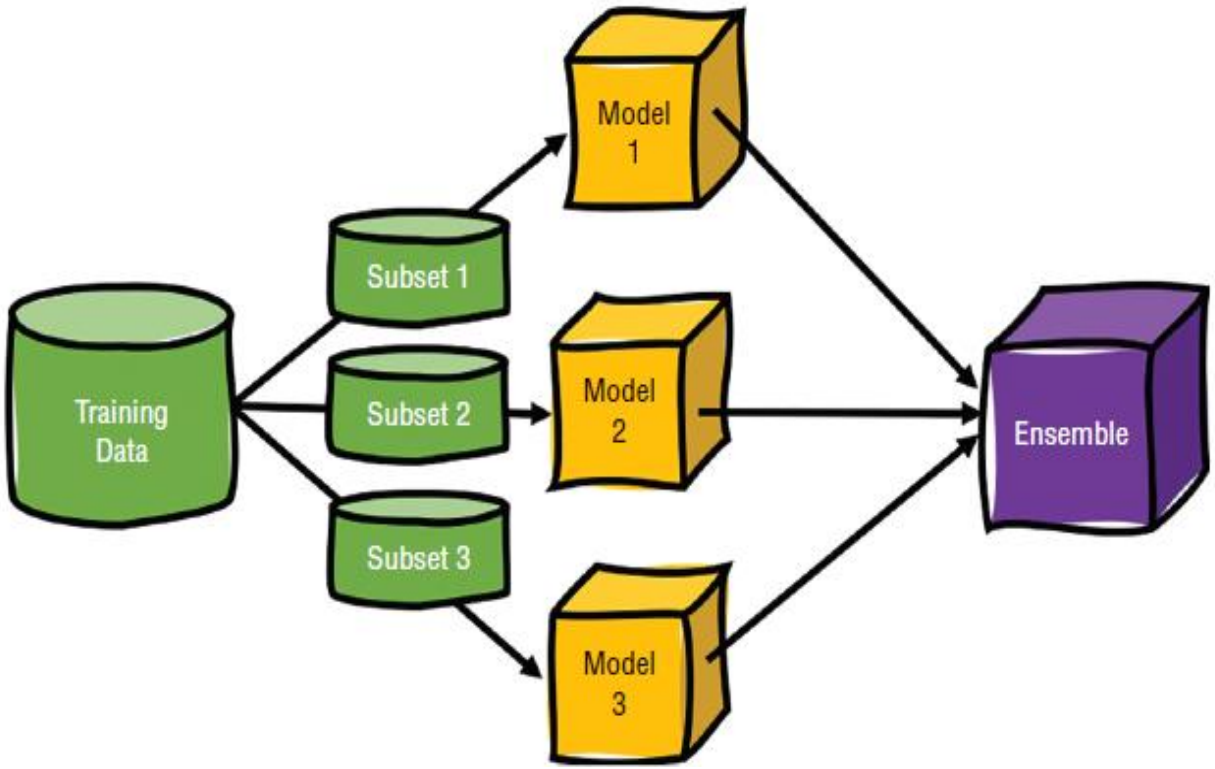
```
def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=5, random_seed=6, n_estimators=10000, learning_rate=0.05, output_process=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, categorical_feature = categorical_feats, free_raw_data=False)
    # parameters
    def lgb_eval(num_leaves, feature_fraction, bagging_fraction, max_depth, min_split_gain, min_child_weight):
        params = {'application':'binary','num_iterations': n_estimators, 'learning_rate':learning_rate, 'early_stopping_round':100, 'metric':'auc'}
        params["num_leaves"] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['min_split_gain'] = min_split_gain
        params['min_child_weight'] = min_child_weight
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, verbose_eval =200, metrics=['auc'])
        return max(cv_result['auc-mean'])
    # range
    lgbBO = BayesianOptimization(lgb_eval, {'num_leaves': (24, 45),
                                            'feature_fraction': (0.1, 0.9),
                                            'bagging_fraction': (0.8, 1),
                                            'max_depth': (5, 8.99),
                                            'min_split_gain': (0.001, 0.1),
                                            'min_child_weight': (5, 50)}, random_state=0)
    # optimize
    lgbBO.maximize(init_points=init_round, n_iter=opt_round)
    # return best parameters
    return lgbBO.res['max']['max_params']

opt_params = bayes_parameter_opt_lgb(X, y, init_round=5, opt_round=10, n_folds=3, random_seed=6, n_estimators=100, learning_rate=0.05)
```

앙상블

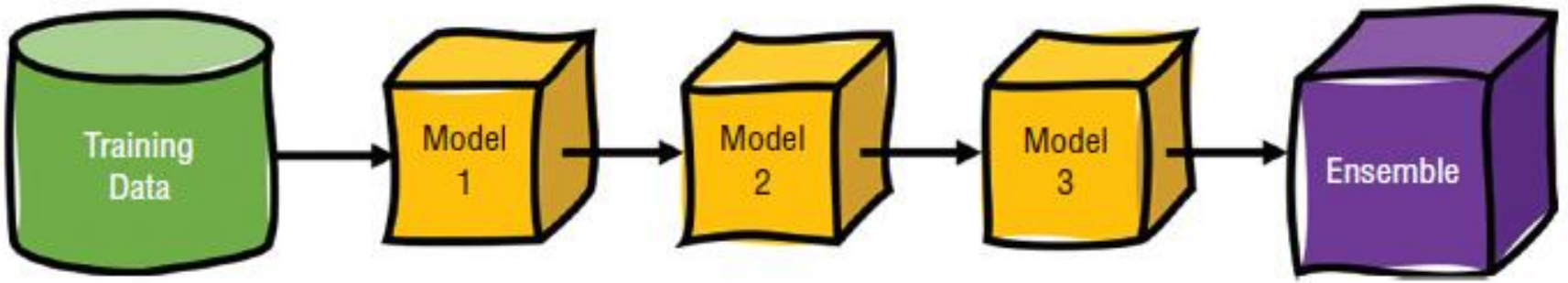
배깅

배깅은 독립적으로 훈련된 동질의 모델을 나타낸다.



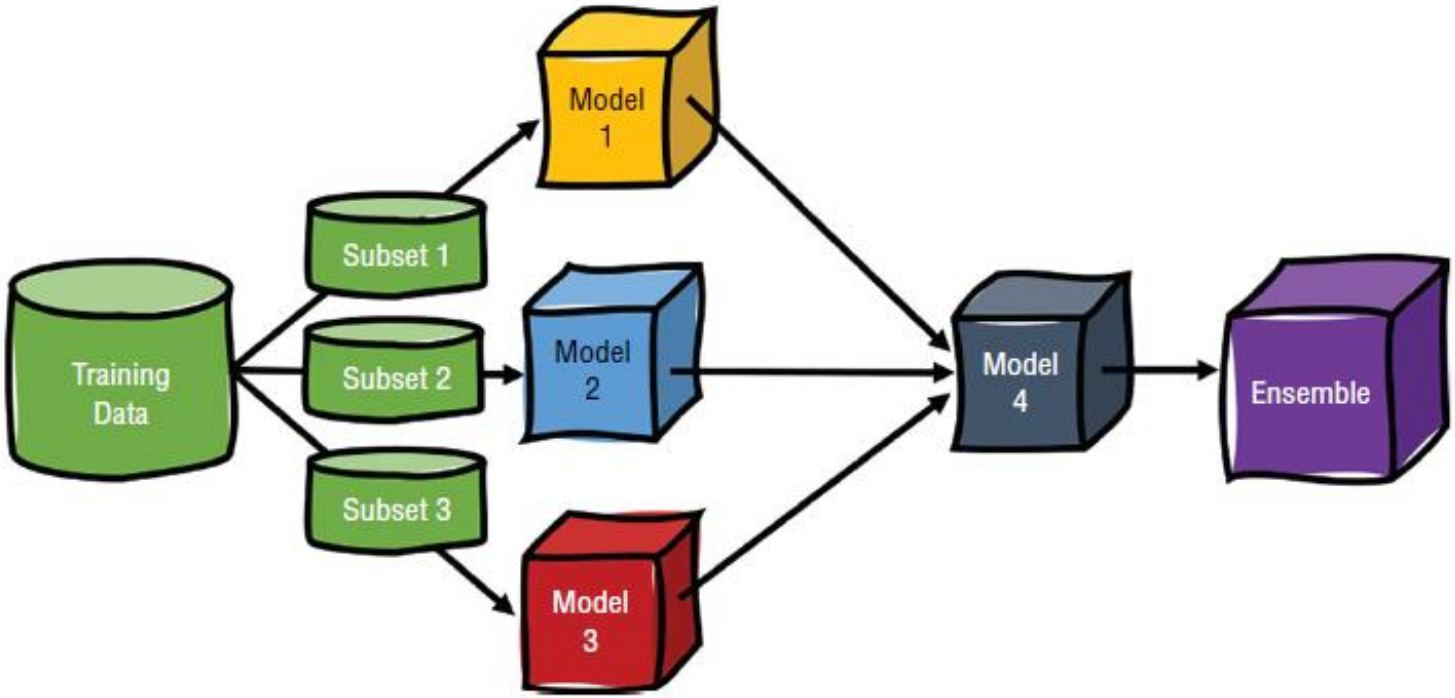
부스팅

부스팅은 동질적인 모델의 순차적으로 적용한다.



스태킹

스태킹은 메타 모델을 이용해 독립적으로 훈련된 모델을 결합한다.



앙상블

□ 단일 모델로 좋은 성과를 내는 것도 중요하지만, 서로 다른 모델의 다양성을 고려하여 결과를 내는 앙상블은 머신러닝의 꽃이며, 다각도로 응용할 수 있다.

□ 여기서는 가장 많이 사용되는 앙상블 3개를 살펴본다.

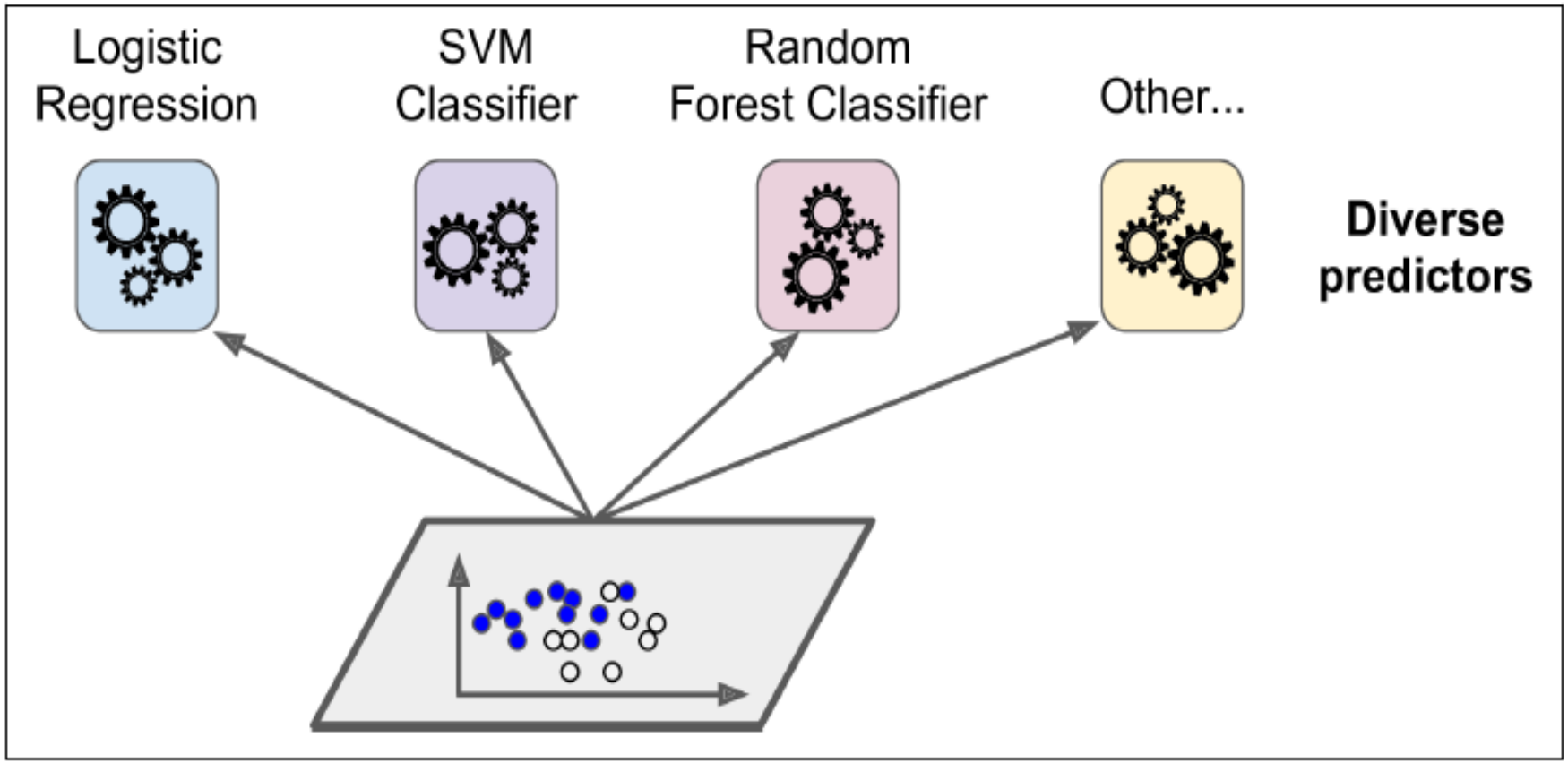
1. Voting Ensemble

2. Average Blendng

3. Stacking

투표기반 분류기

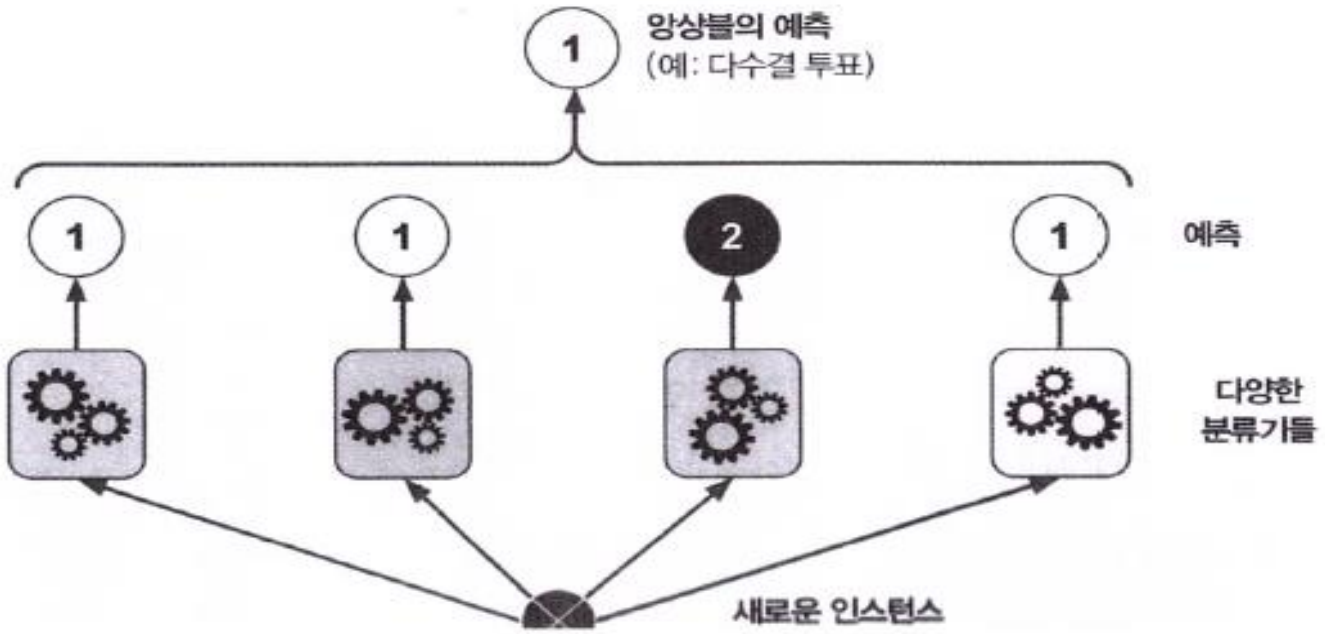
- 여러 분류기 훈련시키기 – 다양한 예측기들



투표기반 분류기

- 직접 투표 분류기의 예측

- 약한 학습기라도 충분히 많고 다양하다면 강한 학습기가 될 수 있다.



- 각자의 모델이 투표를 해 다수결로 클래스로 선택하는 앙상블. Voting 앙상블은 sklearn이 지원해 사용하기 쉽다. Voting Classfier는 sklearn의 ensemble 패키지에 있다.
- 여기서는 Adult 데이터셋으로 연습해본다.

앙상블

□ Voting 앙상블

```
from sklearn.ensemble import VotingClassifier
clfs = [('Logistic Regression', LogisticRegression(random_state=2019)),
        ('Random Forest', RandomForestClassifier(n_estimators=100, max_depth=10, random_state=2019)) ]

vote_clf = VotingClassifier(clfs, voting='soft')
```

```
vote_clf.fit(x_train, y_train)
```

```
y_pred = vote_clf.predict(x_test)
```

```
print('Voting Ensemble Acc : {:.2f}%'.format(vote_clf.score(x_test, y_test)*100))
```

```
Voting Ensemble Acc : 82.51%
```

로지스틱 회귀의 부족한 성능으로 정확도 하락 즉 앙상블도 적당히 괜찮은 모델들이 섞여야 성능이 향상된다.

```
# 단일 모델에서 Random Forest 성능
clf = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=2019)
clf.fit(x_train, y_train)
print('Single Random Forest Acc : {:.2f}%'.format(clf.score(x_test, y_test)*100))
```

```
Single Random Forest Acc : 83.56%
```

앙상블

□ Average 블렌딩

- Average 블렌딩은캐글에서 가장 많이 사용하는 기법이다.
- 회귀의 경우는 모델들이 예측한 값을 n으로 나눠 합치고,
- 분류의 경우에는 각 클래스에 해당하는 확률을 n으로 나눠 합치고 그 중 가장 높은 확률값을 갖는 클래스를 선택한다.

앙상블

□ Average 블렌딩 앙상블

```
val_scores = list()

y_pred = np.zeros_like(y_test, dtype=np.float)

for i, (trn_idx, val_idx) in enumerate(skf.split(x, y)) :
    trn_data, trn_label = x.values[trn_idx, :], y.values[trn_idx]
    val_data, val_label = x.values[val_idx, :], y.values[val_idx]

    # 모델 정의
    clf = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=2019)

    # 모델 학습
    clf.fit(trn_data, trn_label)
    trn_acc = clf.score(trn_data, trn_label)*100
    val_acc = clf.score(val_data, val_label)*100
    print('{} Fold, train Accuracy : {:.2f}%, validation Accuracy : {:.2f}%'.format(i, trn_acc, val_acc))

    val_scores.append(val_acc)
    y_pred += (clf.predict_proba(x_test)[:, 1] / skf.n_splits)

# Mean Validation Score
print('Cross Validation Score : {:.2f}%'.format(np.mean(val_scores)))
```

```
0 Fold, train Accuracy : 83.37%, validation Accuracy : 83.79%
1 Fold, train Accuracy : 84.00%, validation Accuracy : 83.24%
2 Fold, train Accuracy : 83.95%, validation Accuracy : 83.70%
3 Fold, train Accuracy : 83.87%, validation Accuracy : 83.79%
4 Fold, train Accuracy : 83.96%, validation Accuracy : 83.58%
Cross Validation Score : 83.62%
```

로지스틱 회귀의 부족한 성능으로 정확도 하락 즉 앙상블도 적당히 괜찮은 모델들이 섞여야 성능이 향상된다.

```
# 단일 모델에서 Random Forest 성능
clf = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=2019)
clf.fit(x_train, y_train)
print('Single Random Forest Acc : {:.2f}%'.format(clf.score(x_test, y_test)*100))
```

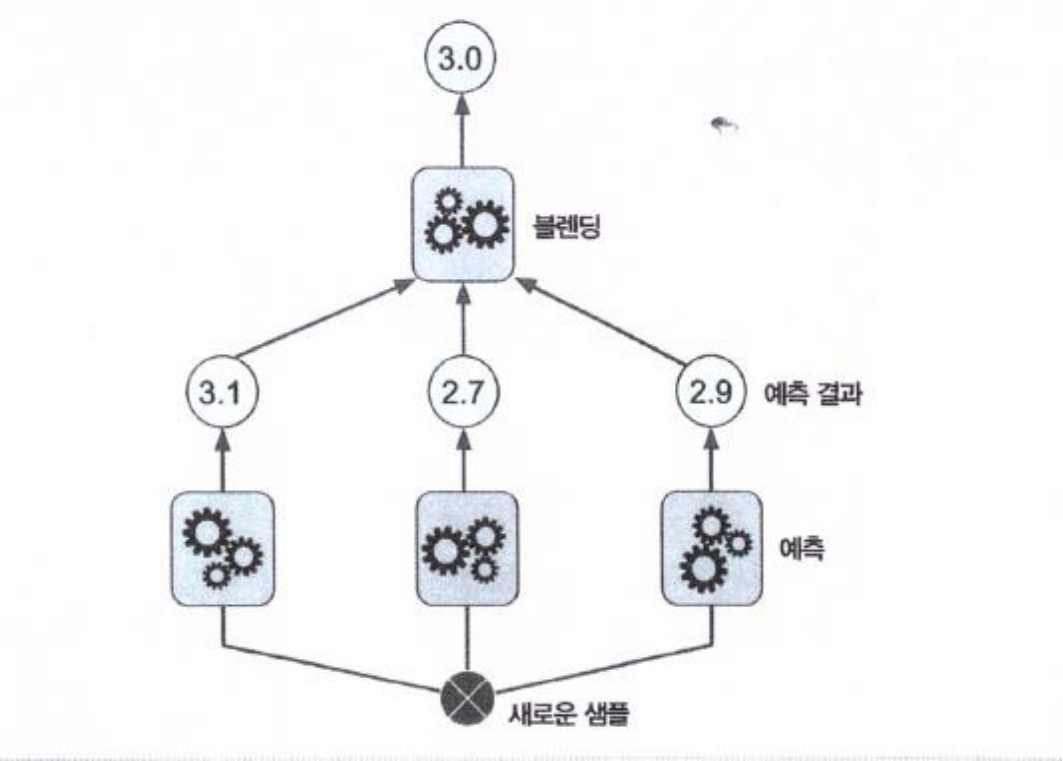
```
Single Random Forest Acc : 83.56%
```

스태킹

스태킹

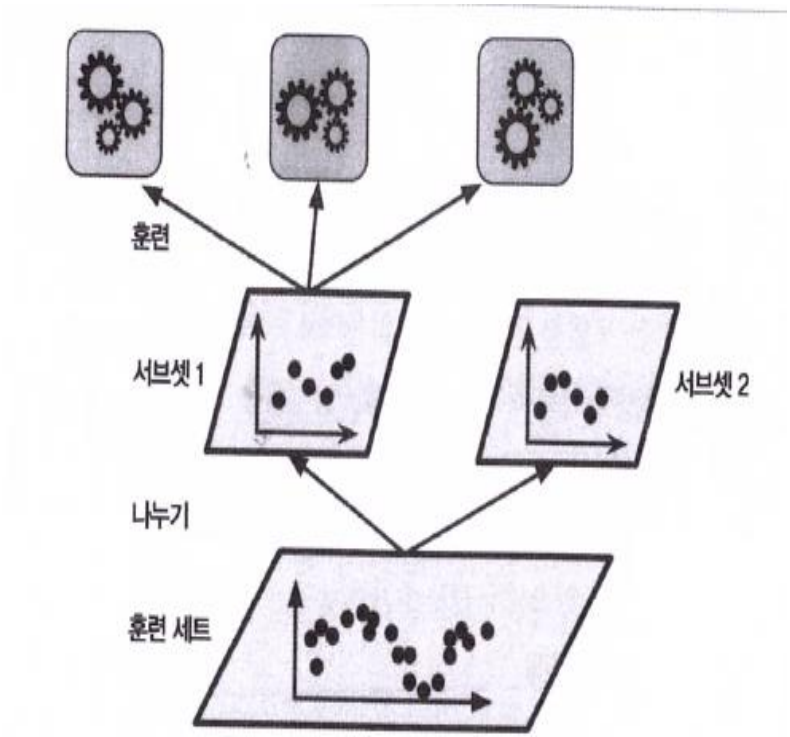
- Stacked Generalization의 줄임말로서 예측기의 예측을 취합하는 모델을 만들어 훈련시키는 것입니다.

- 아래 그림에서 세 예측기가 각각 다른 값 (3.1, 2.7, 2.9)를 예측하고, 마지막 예측기 (블렌더(blender) 또는 메타학습기라고 합니다)가 이들 예측을 입력으로 받아 최종 예측(3.0)을 합니다.

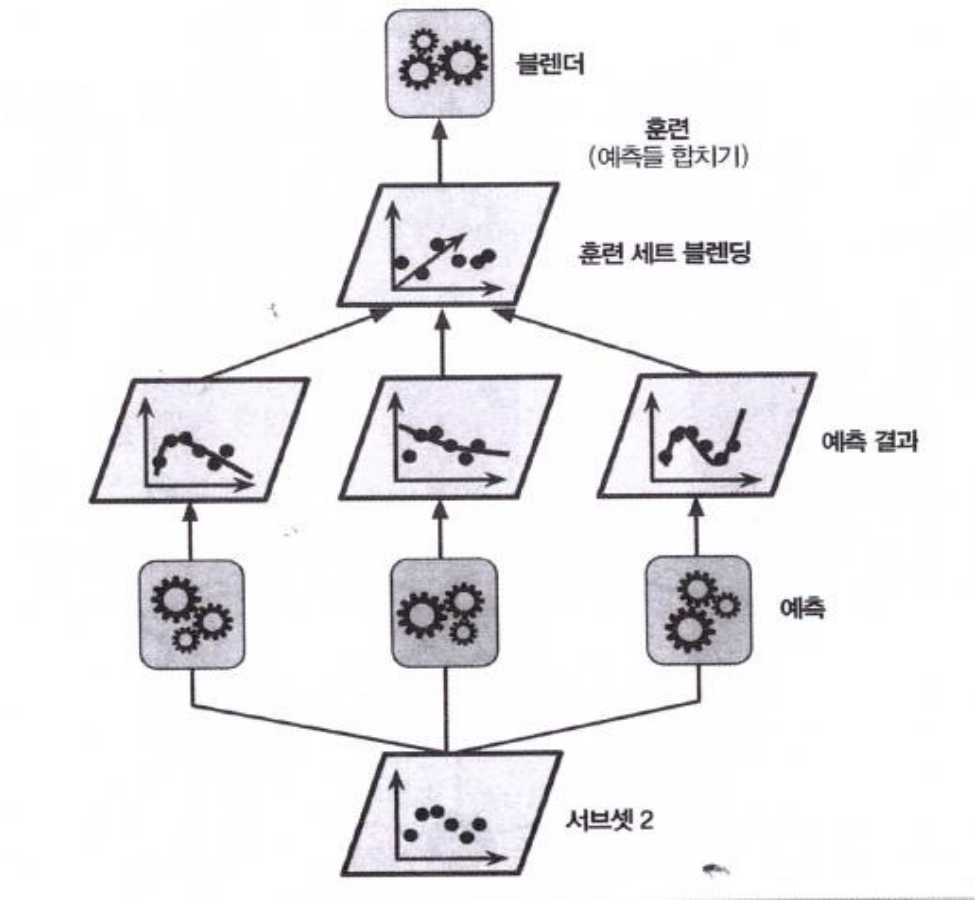


스태킹 훈련 과정

- 첫번째 레이어 훈련하기



블렌더 훈련

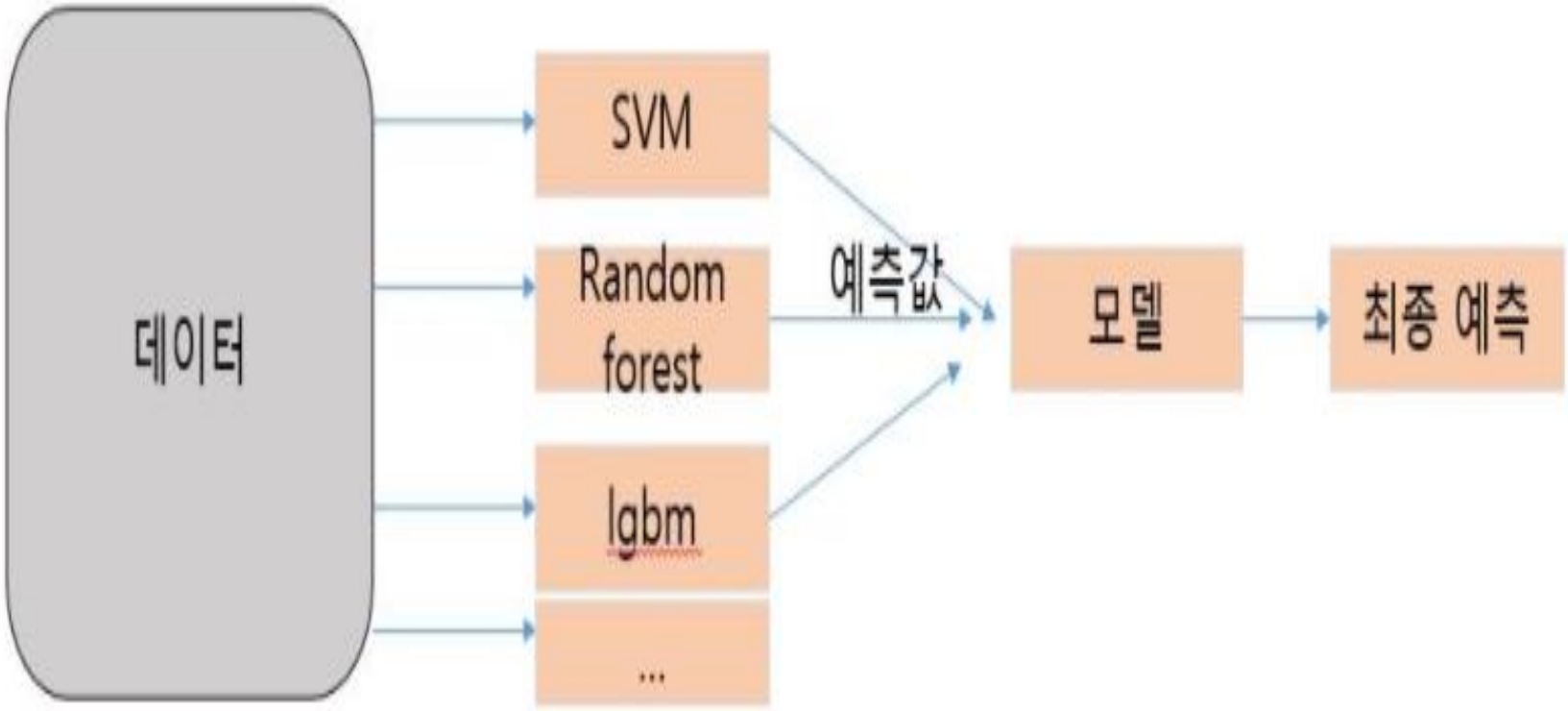


서브셋1(훈련셋)으로 훈련을 첫번째 층을 훈련시키고 서브셋2(홀드아웃셋)에 대한 예측을 만든다.

스태킹 실습

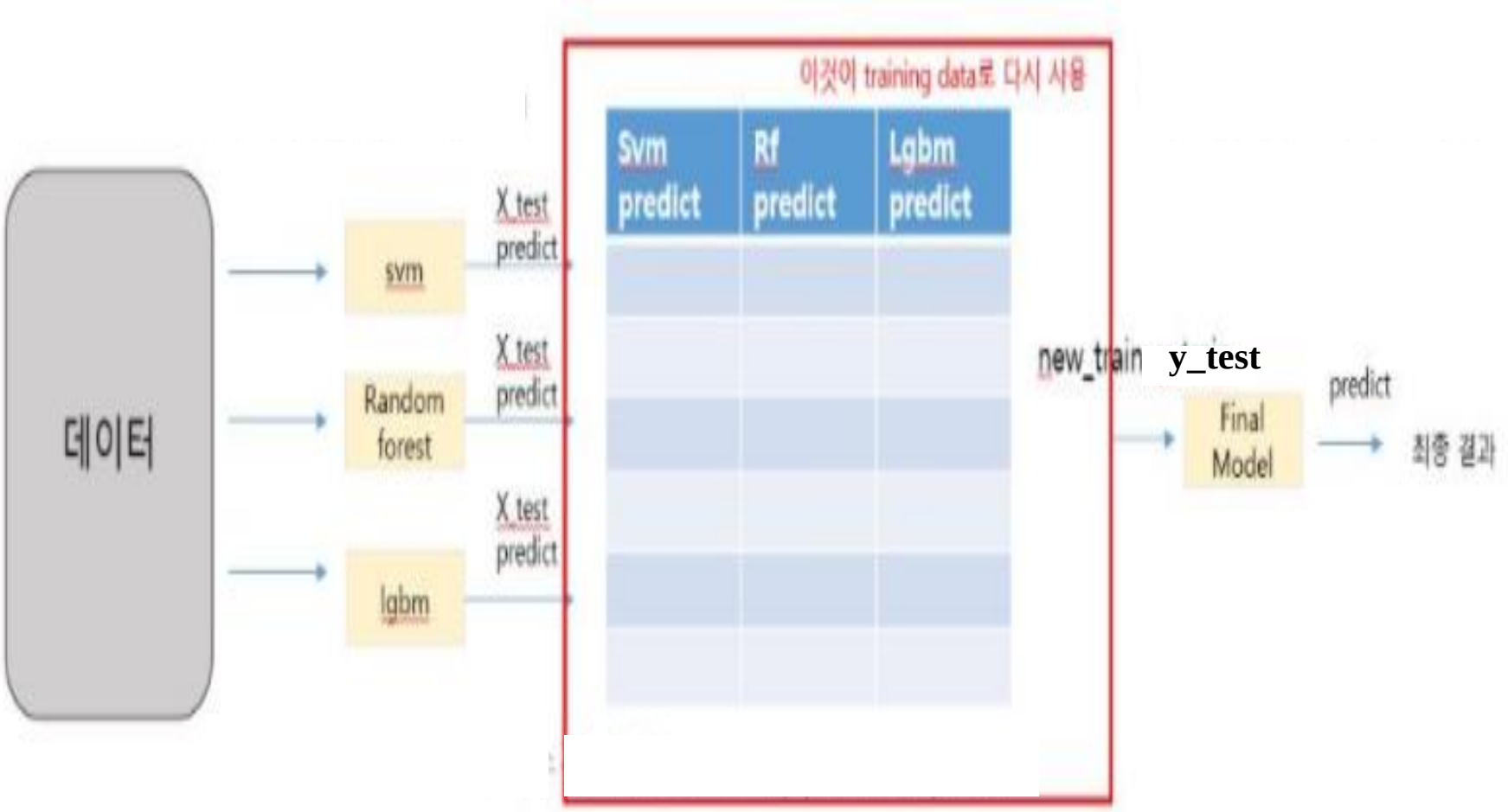
스태킹 앙상블

- 스태킹 앙상블 기본 구조



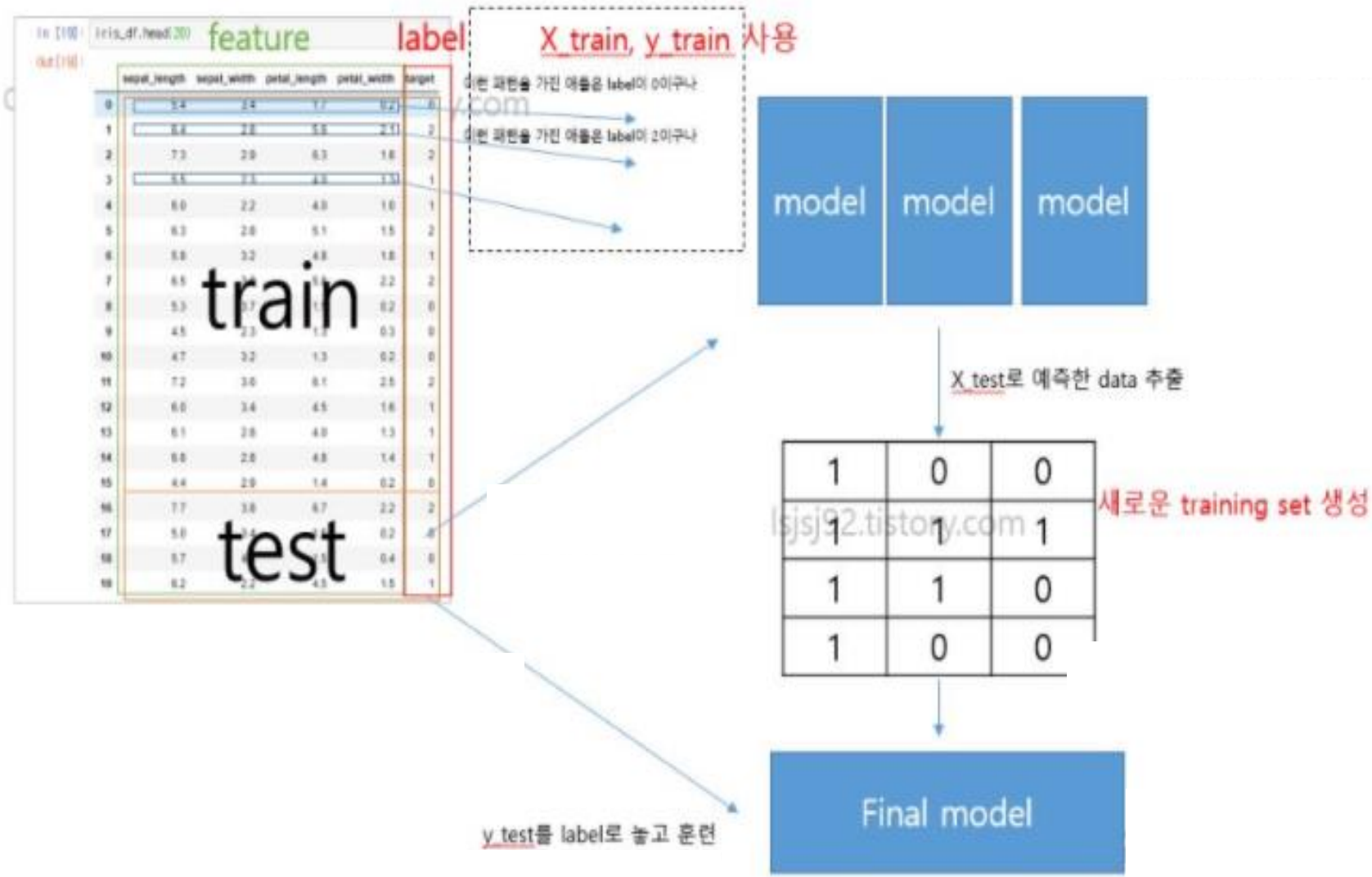
스태킹 앙상블

- 스태킹 앙상블 기본 구조 (세부): 각 훈련셋에서 학습된 예측기의 테스트셋에 대한 예측값을 2단계 예측기의 훈련셋으로 최종 y_test를 예측하도록 학습시킨다.



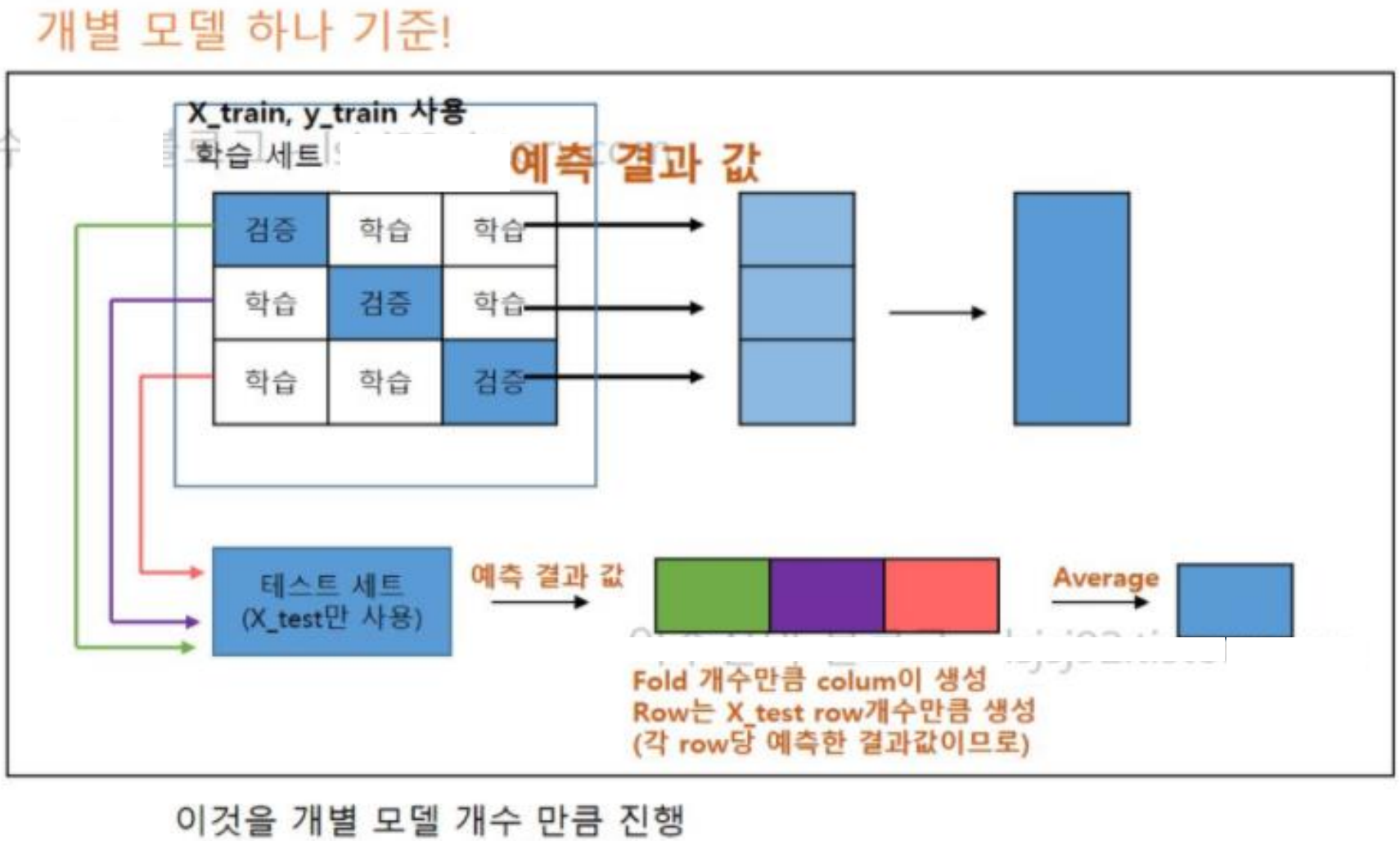
스태킹 앙상블

- 최종 구조


feature
label
X_train, y_train 사용
train
test
model
model
model
X_test로 예측한 data 추출
1 0 0
1 1 1
1 1 0
1 0 0
새로운 training set 생성
Final model
y_test를 label로 놓고 훈련

스태킹 앙상블 – CK 기반

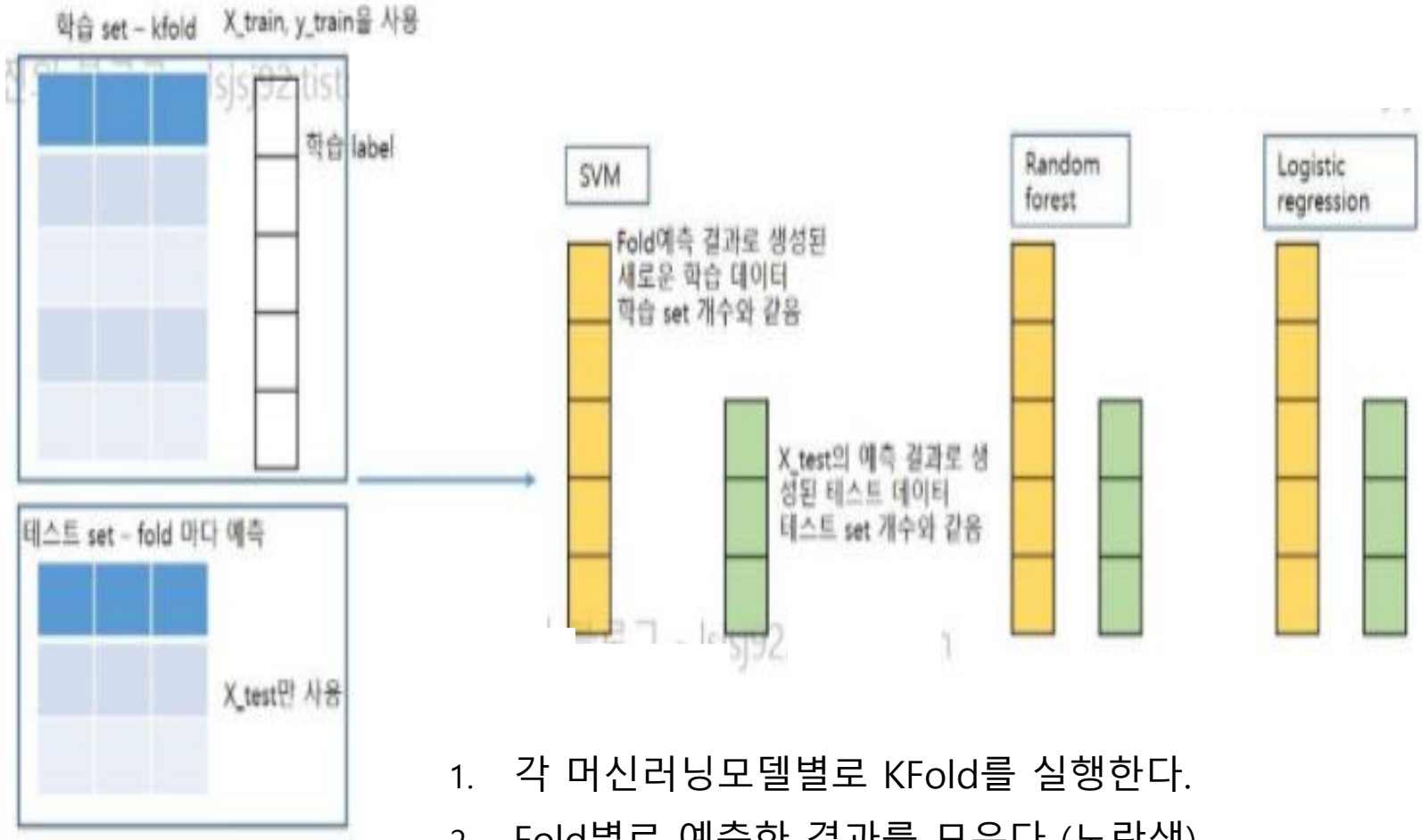
- 하나의 모델로 3겹 실행



이것을 개별 모델 개수 만큼 진행

스태킹 앙상블 – CV 기반

- 각 모델별 K겹 실행: 포인트는 CV를 돌리면 최종 예측값은 원래 X_train과 같은 샘플 크기의 예측치를 얻을 수 있고 이를 y_train과 예측하도록 학습



1. 각 머신러닝모델별로 KFold를 실행한다.
2. Fold별로 예측한 결과를 모은다 (노란색)
3. Fod별로 예측을 수행한 원본 X_test_set을 평균한 결과를 모든다 (초록색)

스태킹 앙상블 – CK 기반

- 스태킹 최종 모델

- Fold validation set으로 예측한 결과를 새로운 new_X_train_set으로 모은다.
- 그리고 X_test_set 으로 예측한 결과를 새로운 new_X_test로 모은다
- 최종적으로 new_X_train_set과 y_train으로 학습한 예측기(lgbm)으로 new_X_test를 사용해 최종 y test를 예측한다.