CONTENTS / 001

Model Selection API : 최적의 머신러닝 모델 찾기

Model Selection 모듈

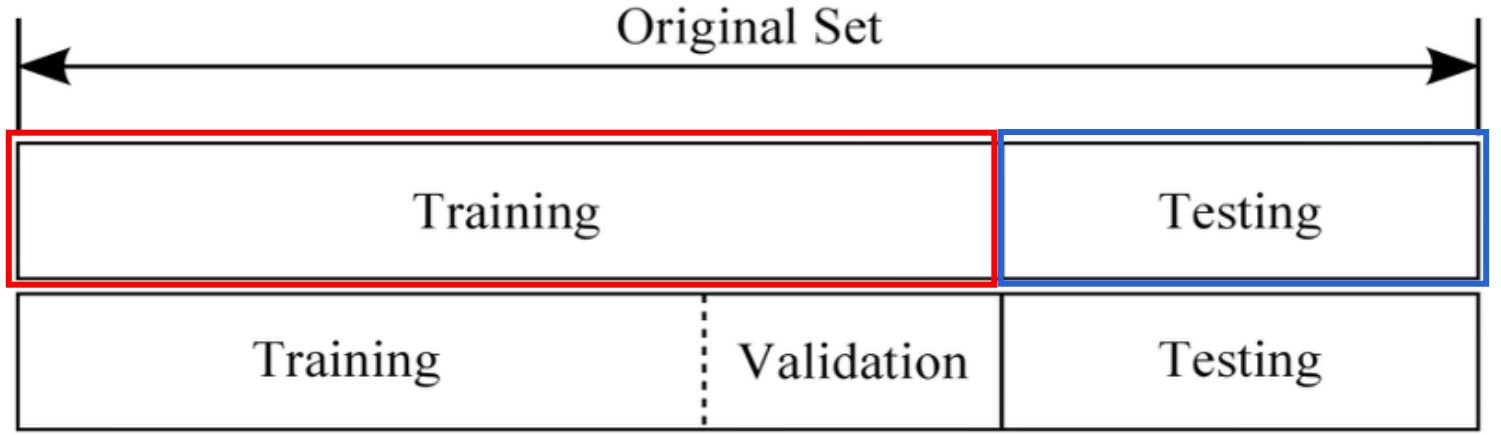
- 모델 구성 및 학습에 필요한 다양한 API 제공
  - 데이터 세트 분리
  - 교차 검증 분할 및 평가
  - 하이퍼 파라미터 튜닝
  - etc.

Model Selection API :최적의 머신러닝 모델 찾기

K-Fold Cross Validation

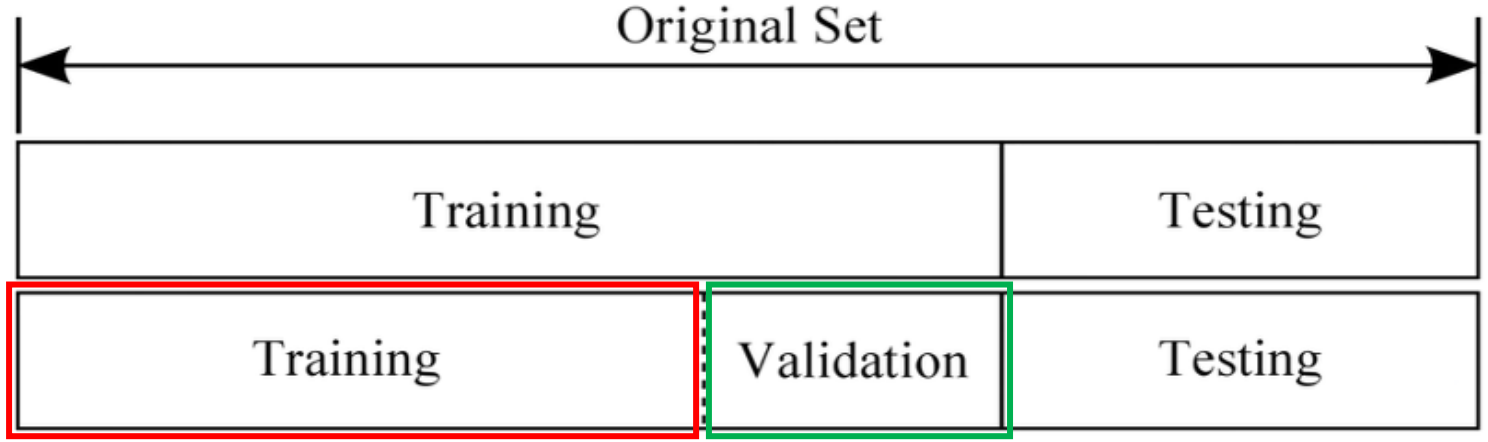
Validation(검증) 데이터 셋

- Training Data Set :
  - 머신러닝 모델의 학습에 사용되는 데이터 셋
- Testing Data Set :
  - 학습된 머신러닝 모델의 서비스 가능 여부를 최종 확인하는 데이터 셋



Validation(검증) 데이터 셋

- Validation Data Set :
  - 학습된 머신러닝 모델을 성능 개선 지표로 사용되는 데이터 셋

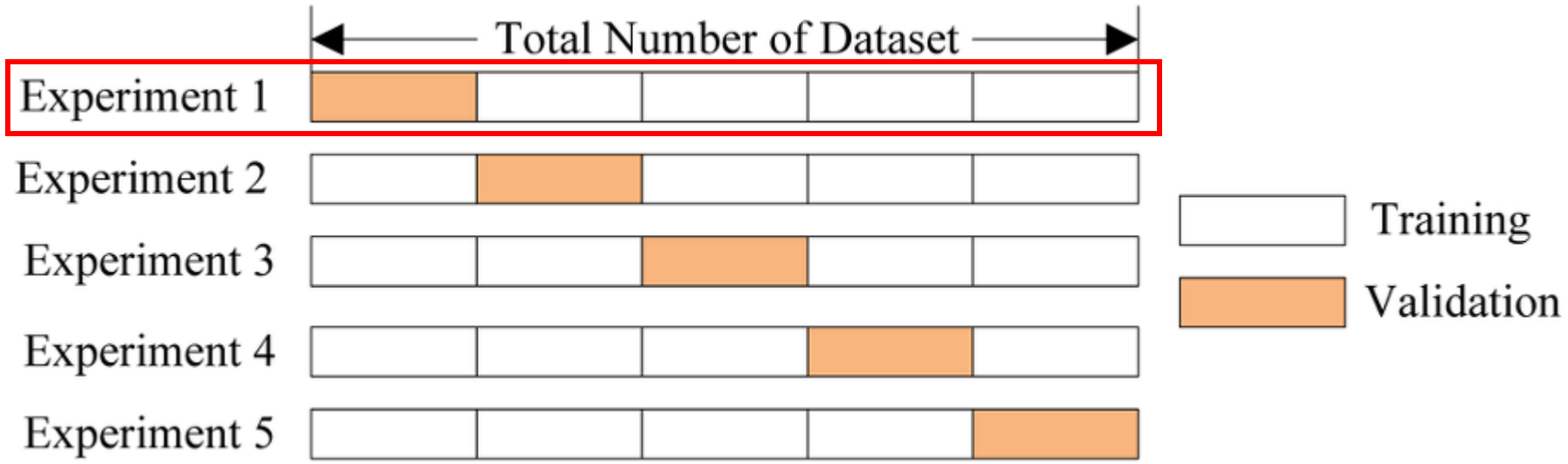


Cross Validation(교차 검증)

- Cross Validation(교차 검증)
  - 테스트 데이터 셋 만을 이용하여 모델의 성능 개선:
    - 테스트 데이터에 만 최적화된 모델이 만들어짐
    - 최종적으로 모델의 서비스 가능 여부를 확인 하는 테스트 데이터의 효과가 사라짐
  - 여러 세트로 구성된 검증 데이터 셋을 통해 성능 개선
    - 좀 더 다양한 데이터에 최적화된 모델로 학습됨
    - 테스트 데이터 셋을 이용하여 모델의 최종 서비스 가능 여부 확인 가능

K Fold Cross Validation(K겹 교차 검증)

- 가장 보편적으로 사용되는 교차검증 방법
- K 개의 데이터 Fold를 만들어 학습과 검증 평가를 반복 수행



K Fold Cross Validation(K겹 교차 검증)

- 데이터 Fold들의 평가 지표를 평균 낸 값이 K Fold 평가 지표
- 사이킷런에서 제공되는 API
  - KFold
  - StratifiedKFold
  - cross_val_score

Stratified K Fold API

- Stratified K Fold Cross Validation
  - 분류분석 :
    - 반드시 Stratified K Fold을 사용한 교차검증을 수행해야 함
  - 회귀분석 :
    - 연속된 숫자 값을 예측하기 때문에 label 데이터의 분포는 의미 없음
    - K Fold를 사용한 Cross Validation 수행

cross_val_score() API

- 교차 검증을 편리하게 수행 할 수 있는 API
- Argument(인자)
  - estimator: 구현하고자 하는 모델(Classification, Regression)
  - X: 데이터 세트
  - y: label 데이터 세트
  - scoring: 검증 지표(성능 평가 지표)
  - cv: Cross Validation의 Fold 숫자
- Return:
  - list 형태의 Fold별 검증 결과(성능지표)

cross_val_score() API

- cross_val_score() API를 이용하여 교차검증 성능 지표 계산
  - 필요 라이브러리 및 Iris 데이터셋 load

Code

```
# K Fold Validation을 위한 cross_val_score() 메서드 로딩
from sklearn.model_selection import cross_val_score

# 모델 구현을 위한 라이브러리 로딩
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
import numpy as np

# load_iris() 메서드를 이용하여 iris 데이터 셋 로드
iris = load_iris()
data = iris.data
label = iris.target
```

cross_val_score() API

- 모델 객체 생성 및, cross_val_score() API 를 이용한 교차검증 성능 지표 계산

Code

```
# DecisionTreeClassifier 모델 객체 생성
dt_clf = DecisionTreeClassifier(random_state=156)

# cross_val_score() 메서드를 이용하여 교차 검증 수행
scores = cross_val_score(estimator=dt_clf,
                         X=data, y=label,
                         scoring='accuracy',
                         cv=3)

# 교차 검증 수행 결과 성능 지표 출력
print('Fold val accuracy:',np.round(scores, 4))
print('Avg val accuracy:', np.round(np.mean(scores), 4))
```

Result

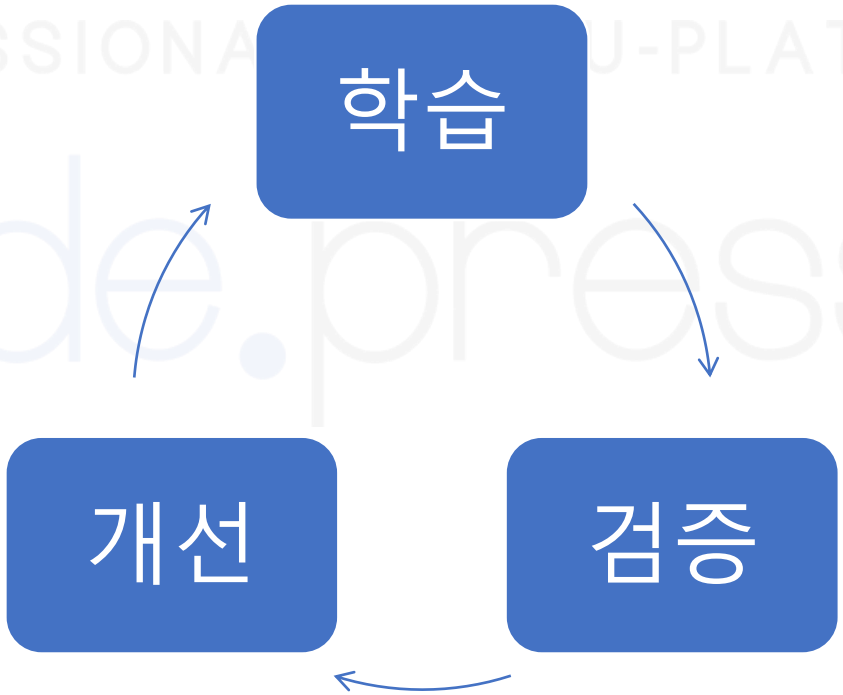
```
Fold val accuracy: [0.98 0.94 0.98]
Avg val accuracy: 0.9667
```

Model Selection API :최적의 머신러닝 모델 찾기

Grid Search CV

하이퍼파라미터 최적화

- 모델의 성능 개선 프로세스
  - 만족할 만한 성능이 나올 때 까지 학습, 검증, 개선 작업을 반복 수행



하이퍼파라미터 최적화

- 하이퍼파라미터
  - 모델을 구성하는 정보 중 데이터를 통해 학습할 수 없는 설정 정보
  - 모델의 검증 결과를 확인 하며 사람이 조정해 줘야 함
    - Decision Tree 의 최대 깊이, leaf node 의 개수, 딥러닝 모델의 구조, etc..

GridSearchCV API

- 모델의 최적 하이퍼 파라미터를 도출하는 API
  - 하이퍼 파라미터:
    - 모델의 주요 구성요소
    - 값 조정을 통해 모델의 성능 개선
  - param_grid 인자로 전달된 파라미터를 순차적으로 적용하여 학습 및 테스트
- Argument(인자)
  - estimator: 구현하고자 하는 모델(Classification, Regression)
  - param_grid: 모델의 튜닝에 사용될 파라미터 정보, 딕셔너리
  - scoring: 검증 지표(성능 평가 지표)
  - cv: Cross Validation의 Fold 숫자
  - refit: 최적의 파라미터로 모델을 재 학습 시킬지 여부

GridSearchCV API

- GridSearchCV API를 이용하여 최적의 모델 학습 시키기
  - 하이퍼 파라미터 최적화를 위한 라이브러리 및 Iris 데이터 셋 load

Code

```
# 하이퍼파라미터 튜닝을 을 위한 GridSearchCV 라이브러리 로딩
from sklearn.model_selection import GridSearchCV

# 모델 구현을 위한 라이브러리 로딩
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

# load_iris() 메서드를 이용하여 iris 데이터 셋 로드
iris = load_iris()
```

GridSearchCV API

- 학습데이터 셋과 테스트 데이터 셋 분할
- 모델 객체 생성 및 검증 대상 파라미터 정보 지정

Code

```
14 # 학습, 테스트 데이터셋 분리
15 x_train, x_test, y_train, y_test = train_test_split(iris.data,
16                                                     iris.target,
17                                                     test_size=0.2,
18                                                     random_state=121)
19
20
21 # DecisionTreeClassifier 모델 객체 생성
22 dtree = DecisionTreeClassifier()
23
24 # 모델의 후보 파라미터 셋(param_grid)을 지정한 딕셔너리 객체 생성
25 parameters = {'max_depth':[1,2,3],
26               'min_samples_split':[2,3]}
```

GridSearchCV API

- 모델 객체와 파라미터 정보를 인자로 전달하여 GridSearchCV 객체 생성
- GridSearchCV 객체의 fit() 메서드를 이용해 학습 및 검증 진행

Code

```
28 # GridSearchCV 객체 생성
29 grid_dtree = GridSearchCV(estimator=dtree,
30                           param_grid=parameters,
31                           cv=3,
32                           refit=True)
33
34 # GridSearchCV 객체의 fit() 메서드를 이용하여
35 # 후보 파라미터 셋의 성능 검증
36 grid_dtree.fit(x_train, y_train)
37
38 # 후보 파라미터 셋의 성능 검증 결과 출력
39 print('Optimal parameter:', grid_dtree.best_params_)
40 print('Max accuracy: {0:.4f}'.format(grid_dtree.best_score_))
```

Result

```
Optimal parameter: {'max_depth': 3, 'min_samples_split': 2}
Max accuracy: 0.9750
```

GridSearchCV API

- 최적의 파라미터로 학습 된 모델의 성능 검증

Code

```
# 최적의 파라미터 모델을 이용하여 예측값 생성
estimator = grid_dtree.best_estimator_
pred = estimator.predict(x_test)

# 최적의 파라미터 모델의 성능지표 출력
print('Test accuracy: {0:.4f}'.format(accuracy_score(y_test,pred)))
```

Result

```
Test accuracy: 0.9667
```