PART Ⅱ

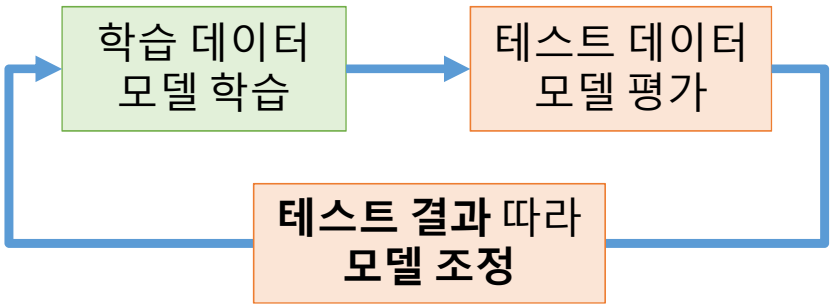
검증과 튜닝

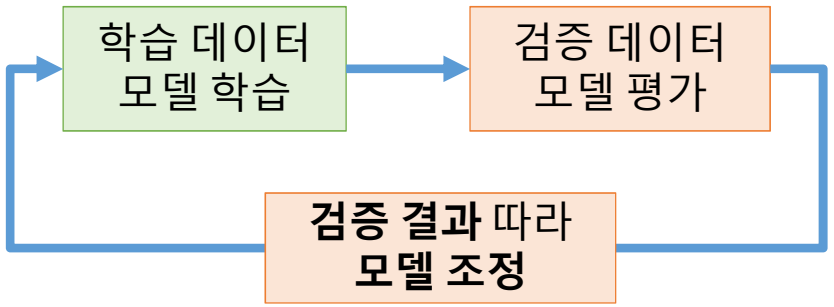
검증(VERIFICATION)

VERIFICATION

◆ 검증 세트

- 학습 중 **모델**을 **평가하기 위한 DataSet**
- **가장 좋은 모델을 선택**할 수 있음
- **일반화가 잘 된 모델**
- 학습 DataSet의 20~30% 정도





VERIFICATION

◆ 교차 검증(CV : Cross Validation)

- 훈련 데이터가 줄어드는 문제 및 **데이터가 충분하지 않은 문제 해결**
- **테스트 데이터에 과대적합(Overfitting)** 문제 해결

- 훈련 데이터를 **동일 크기로 여러 조각 나눈** 후 데이터를 **교차시켜 훈련/검증 데이터**로 활용

[장점] 모든 데이터셋을 훈련과 평가에 활용 가능

=> 정확도 ▲ , 데이터부족 과소적합 방지

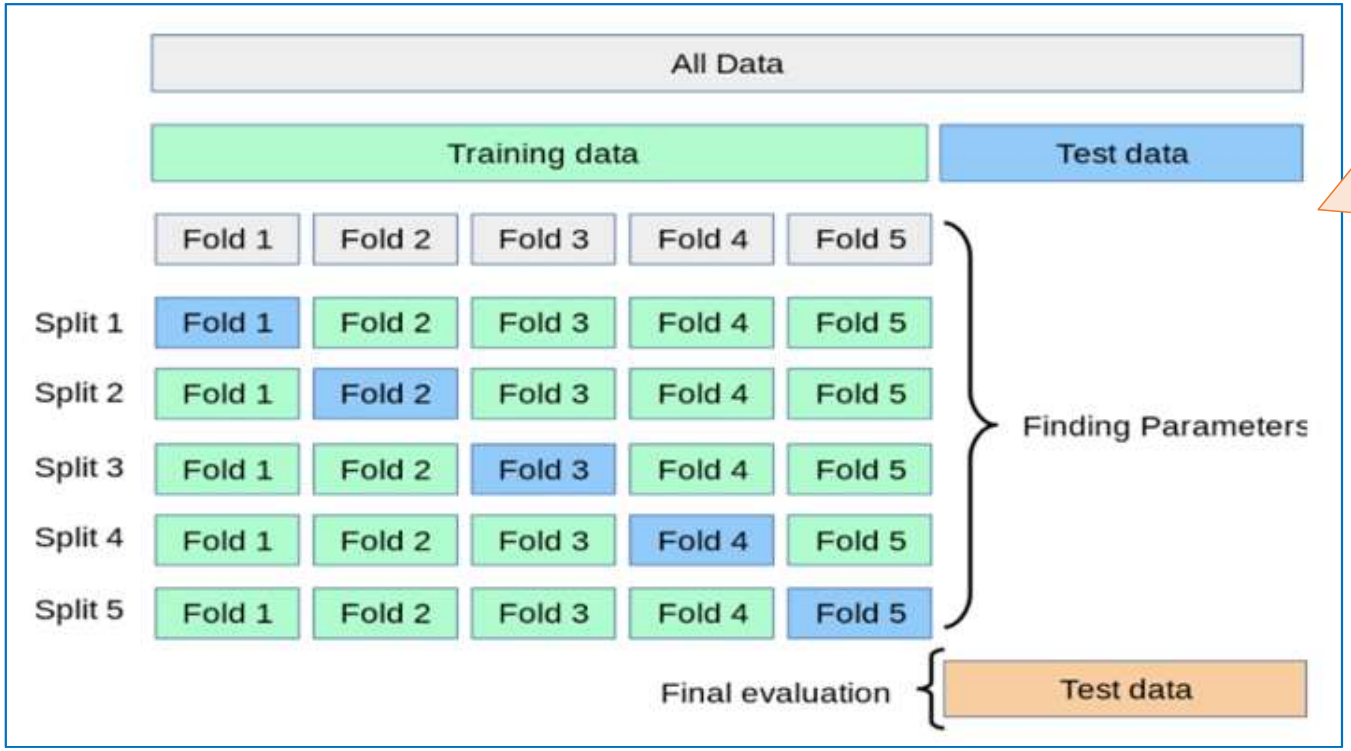
=> 평가용 데이터 편중 ▼, 평가 결과에 일반화된 모델 생성

[단점] 훈련과 평가에 많은 시간 소요

VERIFICATION

◆ 교차 검증(CV : Cross Validation)

- **K-Fold Cross Validation**
  - 가장 일반적으로 사용되는 교차 검증 방법.
  - 보통 **회귀 모델에 사용**, **데이터가 독립적이고 동일한 분포 가진 경우 사용**



총 K개의
성능 평가결과
평균

VERIFICATION

◆ 교차 검증(CV : Cross Validation)

➢ Scikit-Learn LIB

```
sklearn.model_selection. KFold
(*,
  n_splits=5                    : 분할 개수, 최소 2개 이상
  shuffle=False                 : 데이터 섞기 설정
  random_state=None             : 난수 seed 설정
)
```

[단점] Target 데이터가 편중 되는 경우 발생

VERIFICATION

◆ 교차 검증(CV : Cross Validation)

➢ Scikit-Learn LIB

```
# 불균형한 분포도를 가진 레이블 데이터 집합을 위한 KFold 방식
sklearn.model_selection. StratifiedFold
(*,
  n_splits=5                    : 분할 개수, 최소 2개 이상
  shuffle=False                 : 데이터 섞기 설정
  random_state=None             : 난수 seed 설정
)
```

VERIFICATION

◆ 교차 검증(CV : Cross Validation)

➢ Scikit-Learn LIB

sklearn.model_selection.**cross_val_validate**

(estimatr	**: 모델 객체**
X	: 훈련 데이터
y	: 훈련 라벨
groups	: 샘플의 라벨 그룹
cv	: 교차 검증 불할 수 k (기:None)
return_train_score	: 훈련 점수 포함 반환 여부 (기:False)
return_estimator	: 각 학습된 모델 반환여부 (기:False)
)	

[반환] fit_time, score_time, test_score ← Dict 타입 기본 반환값

train_score, estimator ← **parameter 설정**

튜닝(TUNNING)

TUNNING

◆ 튜닝(Tunning)

- 모델 정확도(Accuracy) 높이기 위한 과정들 진행
- **데이터 정제, 여러 가지 모델 테스트**

모델 세부 튜닝

하이퍼파라미터(Hyper-Parameter) 변경하며 모델 테스트 진행

GridSearchCV

RandomizedSearchCV

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

- **모델 성능을 개선**하는 파라미터
  - 과대적합(Overfitting) /과소적합(Unterfitting) 해결
  - 정확도 향상

- 개발자가 지정해야 하는 값
- 모델마다 2~6개 등 개수는 다름

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

LinearRegression

- 특성이 지나치게 많은 경우
- 특성의 일부가 다른 특성들의 조합으로 표현되는 경우 ➔ 상호상관관계 강

➔ 과대적합(Overfitting) 발생 가능성 높음

LinearRegression + Regularization(규제,정규화)

- 특성 선택
- 차원(특성) 축소

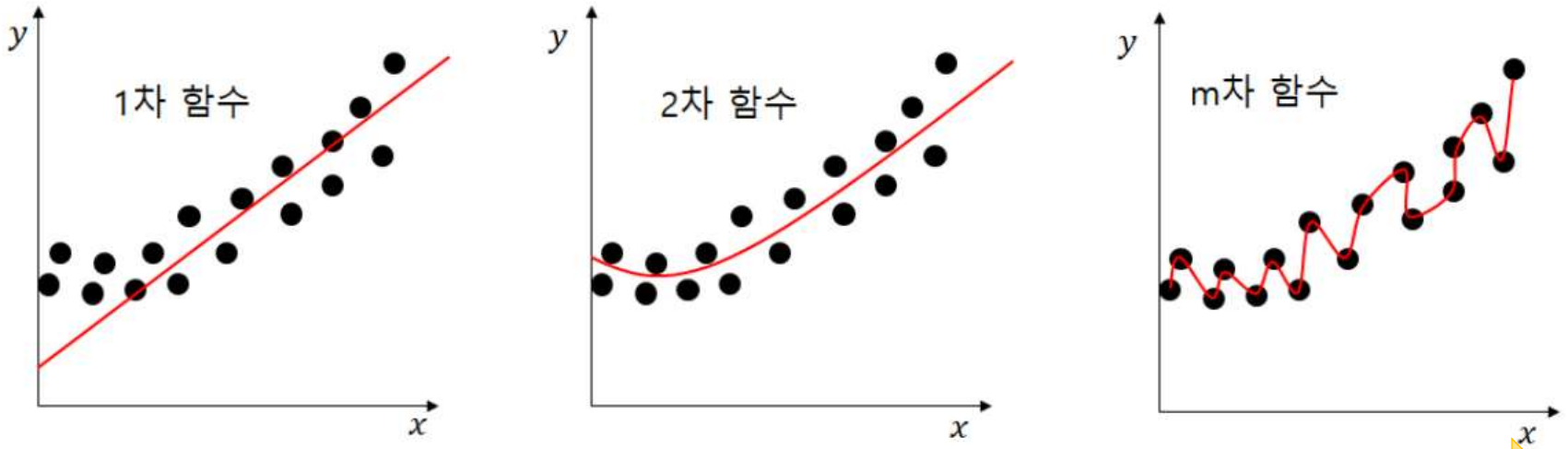
➔ **회귀계수 즉 가중치(w) 크기에 제약 설정**

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

LinearRegression

다항
(고차원)



오버피팅

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

LinearRegression

다중
(고차원)

- 특성 수 증가 ➔ 오버피팅 우려

회귀계수
규제

- 회귀 계수 즉 가중치(w) 값이 매우 커지는 것 방지 ➔ 오버피팅 방지
- 차원 축소와 같은 효과

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

- **RIDGE REGRESSION**
  - 기존 선형 모델에 규제항 추가한 회귀 모델
  - **MSE + 패널티항 => 최소가 되는 가중치와 편향 찾는 모델**
  - 훈련시킬 수록 비용함수가 작아지는 방향으로 진행
  - 비용함수 : **기존 MSE + 가중치 L2 norm 추가**
    - 가중치 제곱 모두 합한 후, 규제 강도 하이퍼파라미터 **alpha 추가**

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

가중치 합 0에 가까워짐

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

➢ Scikit-Learn Lib 사용 - **Ridge 학습객체**

```
from sklearn.linear_model import Ridge
( alpha=1.0,              # 규제 강도 설정, 클수록 회귀계수 작아짐, 양수
 fit_intercept=True     ,  #  절편 상수 사용 여부 설정
 normalize               # 매개변수 무시 여부
 copy_X                  # X의 복사 여부
 max_iter                # 학습 횟수 설정 (기:100)
 tol                     # 정밀도
 solver                  # 계산 알고리즘 (auto, svd,
                           cholesky, lsqr, sparse_cg, sag, saga)
random_state             # 난수 seed 설정  )
```

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

Ridge ← LinearRegression + Regularization(규제,정규화)	
- 가중치(W)의 제곱합을 최소화하는 제약 조건 선형회귀 - 오차를 최소화하는 가중치(W)에 제약 설정 - 특성 크기가 결과에 큰 영향 ➔ 스케일링(Scaling of predictors) 필요	
alpha=1.0	LinearRegression에서 가중치(w)에 대한 규제 설정 범위 : non-negative float [0, inf)

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

- **LASSO REGRESSION**
  - 기존 선형 모델에 규제항 추가한 회귀 모델
  - **MSE + 패널티항 => 최소가 되는 가중치와 편향 찾는 모델**
  - 훈련시킬 수록 비용함수가 작아지는 방향으로 진행
  - 중요도 낮은 가중치 제거함 ➔ **중요한 특성/가중치 선택 훈련**
  - 비용함수 : **기존 MSE + 가중치 L1 norm 추가 (가중치 절대값 합)**

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \sum_{i=1}^{n} | \theta_i |$$

가중치 합 0

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

Lasso (linear model that estimates sparse coefficients with l1 regularization)	
- 가중치(W)의 절대값 합을 오차계산함수에 포함 - 경사하강법 수행 시 가중치(W)가 0이 될 수 있음 - 크기가 결과에 큰 영향 ➔ 스케일링(Scaling of predictors) 필요	
alpha=1.0	LinearRegression에서 가중치(w)에 대한 규제 설정 범위 : non-negative float [0, inf)

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

➢ Scikit-Learn Lib 사용 - Lasso 학습객체

```
from sklearn.linear_model import Lasso
( alpha=1.0,          # 규제 강도 설정, 클수록 회귀계수 작아짐, 양수
 fit_intercept=True   , #  절편 상수 사용 여부 설정
 normalize            # 매개변수 무시 여부
 copy_X               # X의 복사 여부
 max_iter             # 계산 작업 수
 tol                  # 정밀도
warm_start            # 이전 모델을 초기화로 적합하게 사용할 것인지 여부
positive              # 계수가 양수로 사용할 것인지 여부
 solver               # 계산 알고리즘 (auto, svd, cholesky, lsqr, sparse_cg, sag, saga)
random_state          # 난수 seed 설정
selection             # 계수의 업데이트 방법 설정 )
```

TUNNING

◆ 하이퍼파라미터(Hyper-Parameter)

- **Elastic Net REGRESSION**
  - Ridge와 Lasso를 합친 선형 모델
  - Ridge, Lasso 최적화 지점이 다르므로 두 개 값 합쳐서 규제
  - 비용함수 : 기존 MSE + L2 norm + L1 norm

TUNNING

◆ 튜닝(Tunning)

❖ GridSearchCV

- 하이퍼파라미터 튜닝과 교차검증 한번에 처리
  - 분류 => StratifiedFold
  - 회귀 => KFold

TUNNING

◆ 튜닝(Tunning)

➢ Scikit-Learn LIB

sklearn.model_selection. **GridSearchCV**

(estimator

param_grid : dict타입 : 튜닝에 사용할 파라미터

scoring=None : 예측 성능을 측정할 평가 지표 설정

n_jobs=None : 병렬 처리 CPU 수

refit=True : 최적의 하이퍼 파라미터를 찾아 재학습

cv=None : 교차 검증에서 몇개로 분할되는지 지정

return_train_score=False : 학습 점수 반환 여부 설정

)

TUNNING

◆ 튜닝(Tunning)

➢ Scikit-Learn LIB

sklearn.model_selection. **GridSearchCV 속성**

cv_results_	: 튜닝 결과들
best_estimator_	: 튜닝 후 최고의 모델 객체
best_score_	: 튜닝 후 최고 점수
best_params_	: 튜닝 후 최고 파라미터 값
best_index_	: 튜닝 후 최고 모델 인덱스

TUNNING

◆ 튜닝(Tunning)

➢ Scikit-Learn LIB - 예시)

```
# GridSearchCV의 param_grid 설정
params = {  'max_depth': [2, 3],
                    'min_samples_split': [2, 3] }

dtc = DecisionTreeClassifier()
grid_tree = GridSearchCV(dtc, param_grid=params, cv=3, refit=True)
grid_tree.fit(X_train, y_train)

print('best parameters : ', grid_tree.best_params_ )
print('best score : ', grid_tree.best_score_ )
em = grid_tree.best_estimator_
```

TUNNING

◆ 튜닝(Tunning)

➢ Scikit-Learn LIB - 예시)

```
param_grid = {'gamma': [0.0001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(  SVC() ,
                      param_grid=param_grid,
                      scoring=['accuracy'],
                      refit='accuracy',
                      return_train_score=True,
                      cv=3)
grid.fit(X_train, y_train)
```

TUNNING

◆ 튜닝(Tunning)

➢ Scikit-Learn LIB - 예시)

```
lasso = Lasso()
alphas = np.logspace(-4, 0, 200)
parameters = {'alpha': alphas }

lasso_reg = GridSearchCV( lasso, parameters,
                          scoring='neg_mean_squared_error',
                          cv=5 )
lasso_reg.fit(X,y)

print(l asso_reg.best_params_,  lasso_reg.best_score_)
```

TUNNING

◆ 튜닝(Tunning)

➢ Scikit-Learn LIB

sklearn.model_selection.**RandomizedSearchCV**

(estimator

param_distributions : 튜닝에 사용할 파라미터

n_iter=10 : 학습 횟수 지정

scoring=None : 예측 성능을 측정할 평가 방법

n_jobs=None : 병렬 처리 CPU 수

refit=True : 최적의 하이퍼 파라미터를 찾아 재학습

cv=None : 교차 검증에서 몇개로 분할되는지 지정

return_train_score=False : 학습 점수 반환 여부 설정

)