15. 모형진단과 교차검증

1. 모형진단과 교차검증의 이해

○ 과대적합 과소적합

- 모형이 학습데이터에는 정밀도가 높으나 시험데이터에는 정밀도가 현저하게 떨어지면, 이 모형은 학습데이터에 과대적합되었다고 말함.
- 과대적합은 표본수를 늘리거나, 초모수(hyper parameter)를 조정 또는 규제화 등으로 해결할 수 있음.
- 학습데이터의 정밀도가 목표한 정밀도보다 현저하게 낮으면 이 모형은 과소적합되었다고 함.
- 과소적합은 입력변수를 틀리거나 다른 모형을 선택하여 해결하여야 함.

○ 모형진단

- 모형진단은 여러 후보모형 중 가장 좋은 모형을 선택하기 위한 성능비교임.
- 모형진단의 과정은 학습데이터를 통해 모형이 학습되면 이 모형을 검증데이터(validation data)에 적용하여 최적의 초모수를 선택한 후 모형의 성능비교 또는 일반화는 모형의 학습이나 검증에 이용한 적 없는 시험데이터의 정밀도로 비교 점검함.

- 모형진단은 다음 그림과 같이 전체 자료의 분할로부터 출발함.

- 전체 데이터는 학습데이터(Training Set)+검증데이터(Validation Set)+시험데이터 (Test Set)로 구분됨.
- 일반적으로 70:15:15의 비율로 분류하지만 자료의 크기가 매우 큰 빅데이터의 경우 90:5:5의 비율로 분류하기도 함.

○ 초모수 결정

- 초모수는 학습데이터를 이용하여 모형을 학습하고 검증데이터를 통해 조절함.

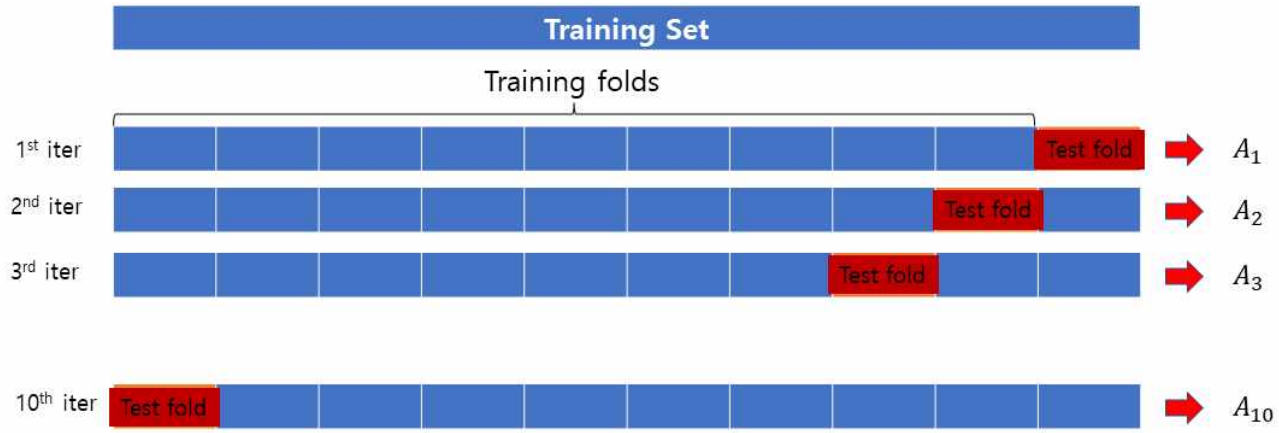
- 예를 들어 SVM에서 결정해야 할 C, 커널함수의 gamma등이 조절해야 할 초모수 중 하나임.
- 학습데이터의 성능을 원하는 수준으로 유지하되, 검증데이터에서의 성능도 학습데이터의 성능과 같거나 최소한 근접하도록 초모수를 조절하고 결정함.
- 즉, 여러개의 초모수 값을 순차적으로 학습데이터와 검증데이터에 적용하여, 검증데이터의 성능이 학습데이터의 성능에 근접하게 하는 초모수를 선택함.

- 하지만 고정된 자료분할은 선택된 검증데이터에 크게 의존하는 치명적인 약점을 가지게 되고, 자료의 크기가 작을 경우 이러한 현상이 두드러짐.
- 즉, 검증데이터가 잘 선택되면 좋은 결과를 보이고 잘못 선택되면 검증데이터의 성능이 나쁘게 나온다면 초모수의 선택이 옳은지 그른지를 판단할 수 없게 됨.
- 이러한 약점을 보완하기 위해 제안된 방법이 k 분할 교차검증 (k-fold cross-validation)임.

○ k-분할 교차검증

- 전체 자료를 학습데이터+시험데이터로 나눈 후, 학습데이터를 임의의 k개 폴드로 분할함. 이 중 k-1개는 학습데이터로, 나머지 1개는 검증데이터로 이용함.
- k개로 분할되어 있으므로 각 분할을 차례대로 검증데이터로 할당하면 총 k번의 서로 다른 검증데이터가 만들어짐.
- 이는 총 k번의 검증데이터의 성능측정이 가능하다는 의미를 가지게 됨.

- 학습데이터를 10등분으로 나누고 초모수 값을 결정하는 절차를 살펴보기로 하겠음.



- 첫 번째 자료분할에서 첫 9개 분할에 속한 데이터를 학습데이터로 하고, 맨 마지막 분할에 있는 데이터를 검증데이터로 사용함.
- 학습데이터를 이용하여 고정된 초모수값 하에서 모형의 모수를 추정하고 이 모형을 검증데이터에 적용하여 구한 성능을 A_1이라고 하겠음.
- 같은 방법으로 10번 반복하게 되면 검증데이터의 성능 $A_1, \ldots, A_{10}$을 구하게 되고, 이들의 평균으로 고정된 초모수의 성능을 평가함.
- 또 다른 초모수로 변경하여 동일한 절차를 진행하여 초모수들 중 가장 우수한 검증성능을 가진 초모수로 모형의 초모수를 결정함.

- k-분할 교차검증에서 분할의 개수는 일반적으로 k=10을 사용하지만 자료가 아주 크면 k=5를 사용하기도 함.
- 자료가 매우 작은 경우에는 k를 자료의 개수로 놓은 1개의 검증데이터 교차검증(leave-one-out cross-validation)을 실시함.

- 일단 초모수가 결정되면 전체 학습데이터를 이용하여 모형의 모수를 다시 추정한 후, 이 모형에 대한 일반화 여부는 시험데이터를 사용하여 점검하게 됨.

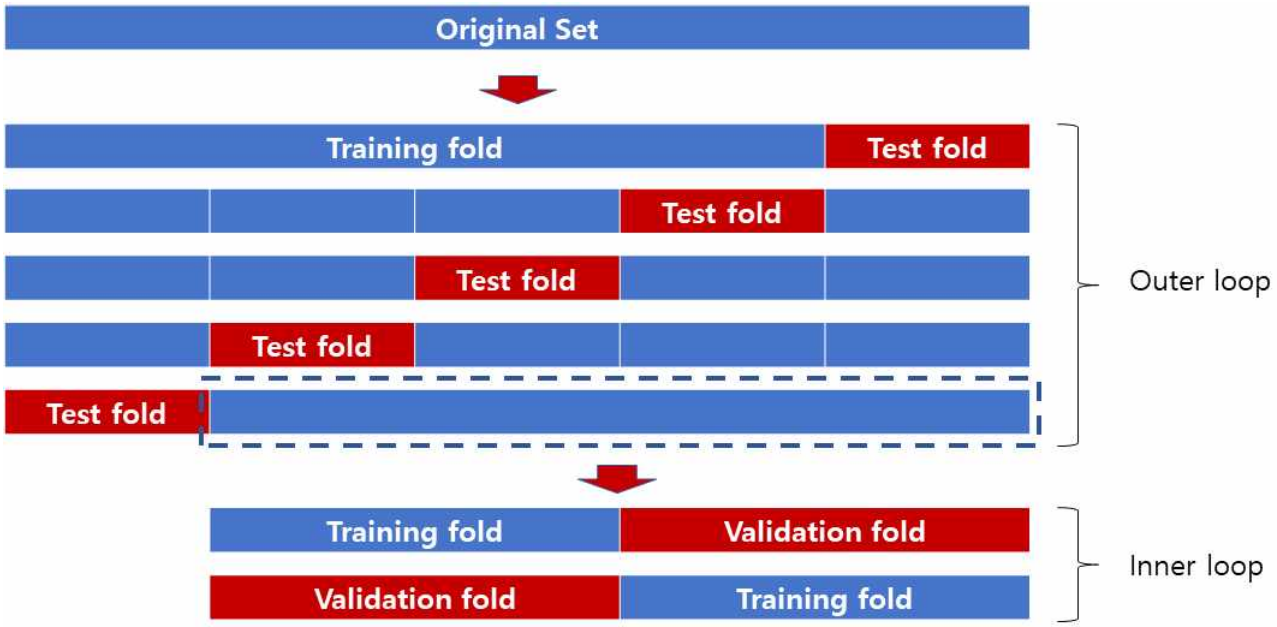
○ 중첩 교차검증(nested cross-validation)

- 중첩 교차검증은 전체 자료를 학습데이터와 시험데이터로 분할 후 다시 학습데이터를 학습데이터와 검증데이터로 구분하여 최적의 초모수 선택 및 모형의 최종성능을 계산하는 방법임.

- 먼저 전체 자료를 학습데이터와 시험데이터를 위한 k_1 분할을 하면 k_1개의 학습데이터 + 시험데이터 세트가 만들어짐.

- 이 세트 각각에 있는 학습데이터를 다시 동일하게 k_2분할을 통해 학습데이터와 검증데이터를 만들어 검증데이터에서의 성능을 구함. 이를 $k_1 - k_2$ 교차검증이라고 함.

- 5-2 교차검증을 통해 살펴보겠음.



- 전체 데이터를 5등분하여 그 중 하나를 교대로 시험데이터로, 나머지 4개를 학습데이터로 사용함.
- 이 학습데이터는 다시 2등분하여 그 중 하나를 교대로 검증데이터로 사용하는 중첩 교차검증임.

- 그러므로 k_1개 분할은 외곽루프를 형성하고, 이 외곽루프 안에는 k_2개의 내부루프(k_2개의 학습데이터+검증데이터 세트)가 형성됨. k_2 내부루프를 이용하여 최적의 초모수를 선택한 후, 외곽루프 시험데이터 모형의 최종성능을 계산함.
- 그러므로 모형의 최종성능 및 일반화의 척도는 k_1개 구해지고 이 k_1개의 성능척도를 평균하여 모형의 최종성능 및 일반화를 점검함.
- 경쟁 모형도 동일한 방법으로 최종성능을 구해 모형간의 성능을 비교할 수 있음.

2. sklearn을 활용한 모형진단과 교차검증

- 모형진단과 교차검증을 실시하기 위해 sklearn 모듈을 활용하여 살펴보도록 하겠음.

- 먼저 앞서 살펴본 바와 같이 전체자료 중 학습데이터를 k 분할 교차검증을 통해서 학습데이터와 검증데이터로 분할하여 분석모델에서 사용하는 초모수들 중 최적의 초모수값을 찾기 위한 방법을 알아보도록 하겠음.
- 분석모델에서 최적의 초모수를 결정하기 위한 k 분할 교차검증을 실시하기 위해서 sklearn.model_selection 모듈의 GridSearchCV().fit() 함수를 이용할 수 있음.

```
from sklearn.model_selection import GridSearchCV
객체명 = GridSearchCV(estimator=분석모델, param_grid=초모수값들,
scoring='accuracy', cv=k분할 교차검증 수).fit(입력변수, 출력변수)
최적 정분류률: 객체명.best_score_
최적 초모수: 객체명.best_params_
```

- GridSearchCV().fit() 함수에 학습데이터 중 입력변수와 출력변수를 차례로 입력함.
- 그리고 estimator 옵션에 분석에 사용되는 분석모델의 함수명을 입력함. 만약 의사결정나무분석을 사용하려 한다면, sklearn.tree 모듈의 DecisionTreeClassifier() 함수를 초모수값들을 입력하지 않은채로 입력하고, SVM을 사용하려 한다면, sklearn.svm 모듈의 SVC() 함수를 초모수값을 입력하지 않은채로 입력함.
- 그리고 param_grid 옵션에 분석모델에서 사용되는 초모수 값들을 딕셔너리 형태로 초모수 옵션 이름과 대입할 값들을 리스트의 형태로 쌍으로 이루어 입력함. 예를 들어 의사결정나무의 최대깊이를 여러개의 값을 두고 확인한다면 {'max_depth':[1, 2, 3, 4, 5]}와 같은 형태로 입력함.
- 만약 초모수의 값들이 여러개라고 한다면, 이러한 딕셔너리를 리스트형태로 묶어서 param_grid 옵션에 입력함.
- 그리고 scoring 옵션에는 최적의 초모수를 결정함에 있어 사용되는 통계량을 결정함. 기본적으로 분석모델에서 score() 함수를 사용하였을 때 출력되는 값이 사용됨.
- 그리고 cv 옵션에는 몇 개의 폴드로 분할할지를 결정함.

- 이렇게 GridSearchCV() 함수를 실행시킨 결과를 객체에 할당한 후 객체에 best_score_함수를 실행하면 최적의 초모수가 사용되었을 때 최적 정분류율이 계산이 됨.
- best_params_ 함수를 실행하면 최적의 초모수값들을 출력하게됨.

- k 분할 교차검증을 통해 구한 최적의 초모수값을 분석모델에 적용하여 학습데이터와 시험데이터의 정분류율을 확인하는 방법을 알아보도록 하겠음.

객체명2 = 객체명.**best_estimator_.fit**(입력변수, 출력변수) 최적초모수적용 학습데이터 정분류율: 객체명2.**score**(입력변수, 출력변수)

- 앞서 GridSearchCV(). 함수를 실행시킨 결과를 할당한 객체에 best_estimator_.fit()함수에 전체 학습데이터를 대입하면 분석모델을 실행시킨 결과와 같음.
- 즉, best_estimator_함수는 GridSearchCV() 함수의 estimator 옵션에 지정한 분석모델에 초모수 값을 지정한 형태임.
- 따라서 best_estimator_.fit() 함수를 실행시킨 결과를 할당한 객체에 score() 함수를 수행하면, 최적의 초모수를 적용한 분석모델의 정분류율을 계산할 수 있음.
- 이 때, 만약 학습데이터의 입력변수와 출력변수를 지정하면 학습데이터의 정분류율이 계산되고, 시험데이터의 입력변수와 출력변수를 지정하면 시험데이터의 정분류율이 계산됨.

- 다음으로 중첩 교차검증을 통해 최적의 초모수를 결정하고 모형을 비교하는 과정을 알아보도록 하겠음.
- 분석모델에서 최적의 초모수를 결정하고 모형을 비교하기 위한 중첩 교차검증을 실시하기 위해서 sklearn.model_selection 모듈의 StratifiedKFold() 함수, GridSearchCV().fit() 함수, cross_val_score() 함수를 이용할 수 있음.

```
from sklearn.model_selection import StratifiedKFold
inner_cv = StratifiedKFold(n_splits=inner loop 횟수, shuffle=True,
random_state=초기난수값)
outer_cv = StratifiedKFold(n_splits=outer loop 횟수, shuffle=True,
random_state=초기난수값)
from sklearn.model_selection import GridSearchCV
객체명 = GridSearchCV(estimator=분석모델, param_grid=초모수값들, scoring='accuracy',
cv=inner_cv)
from sklearn.model_selection import cross_val_score
객체명2 = cross_val_score(객체명, 입력변수, 출력변수, scoring='accuracy', cv=outer_cv)
```

- 먼저 중첩 교차검증은 처음 학습데이터와 시험데이터를 분류하지 않으므로, 최적의 모수를 결정하기 위해 GridSearchCV() 함수의 cv에 inner loop의 수를 직접 입력하게 되면, 자료가 무작위로 추출되어 분석이 이루어지지 않음.
- 즉, k 분할 교차검증 방법에서는 처음 학습데이터와 시험데이터를 분류할 때 사용하는 train_test_split() 함수에서 이미 무작위로 자료를 추출하여 GridSearchCV() 함수의 cv 옵션에 상수값을 대입하였음.
- 하지만, 중첩 교차검증에서는 자료의 무작위 추출을 위해 먼저 StratifiedKFold() 함수를 통해 자료를 출력변수에 따른 층화 무작위 추출을 수행하고자 함.
- StratifiedKFold() 함수의 n_splits 옵션에 중첩 교차검증을 위한 inner loop의 횟수를 입력하고, shuffle 옵션에 True를 반드시 입력함. shuffle 옵션을 통해 자료를 무작위 추출하게 됨.
- 그리고 inner loop를 통해 최적의 모수를 찾기 위해 앞서 사용하였던 GridSearchCV() 함수를 수행하여 객체에 할당함. 이 때, 주의해야할 점은 앞서 k 분할 교차검증에서는 GridSearchCV() 함수에 바로 fit() 함수를 대입하였지만, 중첩 교차검증에서는 GridSearchCV() 함수까지만 사용함.
- 이는, outer loop에서 데이터를 입력하고 여기서 나뉘어진 자료가 초모수 결정에 사용되기 때문임.
- 따라서 GridSearchCV() 함수를 할당한 객체를 outer loop를 수행하여 최종 모형을 평가하는 지표를 산출하는 cross_val_score() 함수에 대입함.
- 그리고 차례로 입력변수와 출력변수를 차례로 대입함. 여기서 사용되는 입력변수와 출력변수 데이터는 전체 데이터임.
- 그리고 cross_val_score() 함수에는 outer loop가 수행되기 때문에 앞

서 StratifiedKFold() 함수를 통해 생성된 outer loop 결과를 cv 옵션에 입력함.

- 이렇게 cross_val_score() 함수를 실행하면 outer loop 마다 최적의 초모수를 결정하고, 이에 따른 정분류률을 계산하여 출력함. 만약 outer loop가 5회 수행되었다고 하면, 총 5개의 정분류률이 계산되어 출력됨.
- 따라서 cross_val_score() 함수를 통해 출력된 정분류률의 평균을 통해 어떠한 분석모델이 최적의 분석모델인지를 판단할 수 있음.

- seaborn 모듈의 iris 데이터에서, species를 예측하기 위해 sepal_length, sepal_width, petal_length, petal_width를 사용하여 k분할 교차검증을 통해 최적의 의사결정나무와 SVM의 모수를 탐색하고, 두 모형을 비교하고자 함.
- info() 함수를 활용하여 자료를 확인한 결과 분석에 사용되는 모든 변수에서 결측자료가 없는 것을 확인할 수 있음.

```
In [1]: import seaborn as sns

In [2]: df = sns.load_dataset('iris')

In [3]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

- 다음으로 모형을 생성하기 위한 Train 데이터와 검증을 위한 Test 데이터를 구분하기 위해 train_test_split() 함수를 사용하였음.

```
In [4]: from sklearn.model_selection import train_test_split

In [5]: train, test = train_test_split(df, test_size=0.3, random_state=1, stratify=df['species'])

In [6]: y_train=train['species']
   ...: X_train=train[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
   ...: y_test=test['species']
   ...: X_test=test[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
```

- 이 때 Train 데이터는 train 객체에 할당하고, Test 데이터는 test 객체에 할당하였음. 그런 후 X_train과 X_test 객체에 train 객체와 test 객체의 입력변수를 선택하여 각각 할당하였고, y_train과 y_test 객체에 train 객체와 test 객체의 출력변수를 선택하여 각각 할당하였음.

- 다음으로 의사결정나무분석의 최적의 초모수를 결정하기 위해 GridSearchCV() 함수를 사용하였음.

```
In [7]: from sklearn.model_selection import GridSearchCV
   ...: from sklearn.tree import DecisionTreeClassifier

In [8]: GS1_1 = GridSearchCV(estimator=DecisionTreeClassifier(random_state=1),
   ...:                      param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'criterion':['gini']},
   ...:                                  {'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'criterion': ['entropy']}],
   ...:                      scoring='accuracy', cv=5).fit(X_train, y_train)
```

- 의사결정나무분석의 초모수인 나무의 최대깊이와 불순도측도를 각각 설정하였음. 나무의 최대깊이는 1~7까지 단계적인 값과 최대깊이를 설정하지 않은 경우를 설정하였고, 불순도 측도는 gini 지수와 entropy 지수 두가지를 각각 사용하였음. 그리고 cv옵션에 5를 입력하여 5분할 형태로 자료를 분할하였음.

```
In [9]: print(GS1_1.best_score_)
0.9333333333333332

In [10]: print(GS1_1.best_params_)
{'criterion': 'gini', 'max_depth': 2}
```

- best_score_ 함수와 best_params_ 함수를 통해 최적의 초모수와 정분류률을 확인한 결과 gini지수와 최대깊이를 2로 하였을 때 정분류률이 0.93333으로 가장 높게 나타나는 것을 확인할 수 있음.

```
In [11]: clf1 = GS1_1.best_estimator_.fit(X_train, y_train)

In [12]: print(clf1.score(X_train,y_train))
0.9523809523809523

In [13]: print(clf1.score(X_test, y_test))
0.9555555555555556
```

- 이러한 초모수 값을 이용하여 학습데이터의 정분류률을 score() 함수를 통해 계산하였더니 0.9523809로 나타났고, 시험데이터의 정분류률은 0.9555555로 나타났음.
- 학습데이터의 정분류률와 시험데이터의 정분류률이 비슷한 값을 나타나는 것을 통해 모형이 안정화되어 있다는 것을 알 수 있음.

- 다음으로 SVM의 최적의 초모수를 결정하기 위해 GridSearchCV() 함수를 사용하였음.

```
In [14]: from sklearn.model_selection import GridSearchCV
    ...: from sklearn.svm import SVC

In [15]: param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]

In [16]: param_grid=[{'C': param_range, 'kernel': ['linear']},
    ...:             {'C': param_range, 'gamma': param_range, 'kernel': ['rbf']}]

In [17]: GS2_1 = GridSearchCV(estimator=SVC(random_state=1),
    ...:                      param_grid=param_grid,
    ...:                      scoring='accuracy', cv=5).fit(X_train, y_train)
```

- SVM의 초모수인 C, gamma, kernel을 각각 설정하였음. C와 gamma는 0.0001에서 1000까지 한 단위씩 증가하도록 를 설정하였고, kernel은 linear와 rbf 두 가지를 각각 사용하였음. 그리고 cv옵션에 5를 입력하여 5분할 형태로 자료를 분할하였음.

```
In [18]: print(GS2_1.best_score_)
0.9619047619047618

In [19]: print(GS2_1.best_params_)
{'C': 1.0, 'gamma': 1.0, 'kernel': 'rbf'}
```

- best_score_ 함수와 best_params_ 함수를 통해 최적의 초모수와 정분류률을 확인한 결과 C와 gamma가 1, kernel을 rbf로 하였을 때 정분류률이 0.96190476으로 가장 높게 나타나는 것을 확인할 수 있음.

```
In [20]: clf2 = GS2_1.best_estimator_.fit(X_train, y_train)

In [21]: print(clf2.score(X_train,y_train))
0.9809523809523809

In [22]: print(clf2.score(X_test, y_test))
0.9777777777777777
```

- 이러한 초모수 값을 이용하여 학습데이터의 정분류률을 score() 함수를 통해 계산하였더니 0.980595238로 나타났고, 시험데이터의 정분류률은 0.97777777로 나타났음.
- 학습데이터의 정분류률와 시험데이터의 정분류률이 비슷한 값을 나타나는 것을 통해 모형이 안정화되어 있다는 것을 알 수 있음.
- 또한, SVM의 경우 의사결정나무에 비해 정분류률이 높은 것으로 보아 의사결정나무분석에 비해 SVM이 더 효과적인 분석모델이라는 것을 알 수 있음.

- 다음으로 중첩 교차검증 방법을 통해 의사결정나무분석의 최적의 초모수

를 결정하고 모형을 평가해보도록 하겠음.

- 먼저 자료를 무작위 층화 추출을 하기 위해 StratifiedKFold() 함수를 사용하였음.

```
In [26]: X=df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
    ...: y=df['species']

In [27]: from sklearn.model_selection import StratifiedKFold

In [28]: inner_cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=1)

In [29]: outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
```

- 중첩 교차검증 방법은 전체 데이터를 사용하여 Outer loop와 Inner loop를 설정하므로 전체 데이터를 각각 입력변수 X와 출력변수 y에 할당하였음.
- 그리고 inner loop는 3회, outer loop는 5회 분할하기 위해 n_splits 옵션에 각각 3과 5를 입력하였음. 그리고 무작위 추출을 위해 shuffle 옵션에 True를 입력하였음.

- 다음으로 inner loop를 통해 의사결정나무분석의 최적의 초모수를 결정하기 위해 GridSearchCV() 함수를 사용하였음.

```
In [30]: from sklearn.model_selection import GridSearchCV
    ...: from sklearn.tree import DecisionTreeClassifier

In [31]: GS1_2 = GridSearchCV(estimator=DecisionTreeClassifier(random_state=1),
    ...:                      param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'criterion':['gini']},
    ...:                                  {'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'criterion': ['entropy']}],
    ...:                      scoring='accuracy', cv=inner_cv)
```

- 의사결정나무분석의 초모수인 나무의 최대깊이와 불순도측도를 각각 설정하였음. 나무의 최대깊이는 1~7까지 단계적인 값과 최대깊이를 설정하지 않은 경우를 설정하였고, 불순도 측도는 gini 지수와 entropy 지수 두가지를 각각 사용하였음. 그리고 cv옵션에 StratifiedKFold() 함수를 통해 생성한 inner_cv 객체를 입력하였음.

- 다음으로 outer loop를 통해 의사결정나무분석의 정분류률을 계산하기 위해 cross_val_score() 함수를 사용하였음.

```
In [32]: from sklearn.model_selection import cross_val_score,cross_validate

In [33]: scores = cross_val_score(GS1_2, X, y, scoring='accuracy', cv=outer_cv)

In [34]: print(scores)
[0.93333333 1.         0.93333333 0.96666667 0.9        ]

In [35]: from numpy import mean, std

In [36]: print('CV accuracy: %.3f +/- %.3f' % (mean(scores), std(scores)))
CV accuracy: 0.947 +/- 0.034
```

- 5회에 걸친 outer loop를 통해 나타난 정분류률이 각각 0.933, 1.000, 0.933, 0.9666, 0.9000으로 나타났음. 이를 평균과 분산을 구하기 위해 numpy 모듈의 mean과 std 함수를 사용하여 계산한 결과 중첩 교차검증을 통해 나타난 의사결정나무분석의 정분류률은 평균 0.947, 표준편차 0.034가 나타났음.

- 다음으로 inner loop를 통해 SVM의 최적의 초모수를 결정하기 위해 GridSearchCV() 함수를 사용하였음.

```
In [37]: from sklearn.model_selection import GridSearchCV
    ...: from sklearn.svm import SVC

In [38]: param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]

In [39]: param_grid=[{'C': param_range, 'kernel': ['linear']},
    ...:             {'C': param_range, 'gamma': param_range, 'kernel': ['rbf']}]

In [40]: GS2_2 = GridSearchCV(estimator=SVC(random_state=1),
    ...:                      param_grid=param_grid,
    ...:                      scoring='accuracy', cv=inner_cv)
```

- SVM의 초모수인 C, gamma, kernel을 각각 설정하였음. C와 gamma는 0.0001에서 1000까지 한 단위씩 증가하도록 를 설정하였고, kernel은 linear와 rbf 두 가지를 각각 사용하였음. 그리고 cv옵션에 StratifiedKFold() 함수를 통해 생성한 inner_cv 객체를 입력하였음.

- 다음으로 outer loop를 통해 SVM의 정분류률을 계산하기 위해 cross_val_score() 함수를 사용하였음.

```
In [41]: from sklearn.model_selection import cross_val_score,cross_validate

In [42]: scores = cross_val_score(GS2_2, X, y, scoring='accuracy', cv=outer_cv)

In [43]: print(scores)
[1.         1.         0.93333333 1.         0.93333333]

In [44]: from numpy import mean, std

In [45]: print('CV accuracy: %.3f +/- %.3f' % (mean(scores), std(scores)))
CV accuracy: 0.973 +/- 0.033
```

- 5회에 걸친 outer loop를 통해 나타난 정분류률이 각각 1.000 1.000, 0.933, 1.000, 0.9333으로 나타났음. 이를 평균과 분산을 구하기 위해 numpy 모듈의 mean과 std 함수를 사용하여 계산한 결과 중첩 교차검증을 통해 나타난 SVM의 정분류률은 평균 0.973, 표준편차 0.033이 나타났음.
- 중첩 교차검증을 통해 SVM의 경우 의사결정나무에 비해 정분류률이 높은 것으로 보아 의사결정나무분석에 비해 SVM이 더 효과적인 분석모델이라는 것을 알 수 있음.

3. 모형진단과 교차검증의 실습

※ seaborn 모듈의 penguins 데이터에서, species를 예측하기 위해 bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g을 사용하여 k 분할 교차검증을 통해 최적의 의사결정나무와 SVM의 모수를 탐색하고, 두 모형을 비교해 보세요.

- info() 함수를 활용하여 자료를 확인한 결과 bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g에서 결측자료가 존재하는 것을 확인할 수 있음. 따라서 dropna() 함수를 이용하여 결측값을 제거할 필요가 있음.

```
In [1]: import seaborn as sns

In [2]: df = sns.load_dataset('penguins')

In [3]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB

In [4]: df = df.dropna(subset=['species', 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'],
   ...:                how='any', axis=0)
```

- 다음으로 모형을 생성하기 위한 Train 데이터와 검증을 위한 Test 데이터를 구분하기 위해 train_test_split() 함수를 사용하였음.

```
In [5]: from sklearn.model_selection import train_test_split
   ...: train, test = train_test_split(df, test_size=0.3, random_state=1, stratify=df['species'])

In [6]: y_train=train['species']
   ...: X_train=train[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']]
   ...: y_test=test['species']
   ...: X_test=test[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']]
```

- 이 때 Train 데이터는 train 객체에 할당하고, Test 데이터는 test 객체에 할당하였음. 그런 후 X_train과 X_test 객체에 train 객체와 test 객체의 입력변수를 선택하여 각각 할당하였고, y_train과 y_test 객체에 train 객체와 test 객체의 출력변수를 선택하여 각각 할당하였음.

- 다음으로 의사결정나무분석의 최적의 초모수를 결정하기 위해 GridSearchCV() 함수를 사용하였음.

```
In [7]: from sklearn.model_selection import GridSearchCV
   ...: from sklearn.tree import DecisionTreeClassifier

In [8]: GS1_1 = GridSearchCV(estimator=DecisionTreeClassifier(random_state=1),
   ...:                      param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'criterion':['gini']},
   ...:                                  {'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'criterion': ['entropy']}],
   ...:                      scoring='accuracy', cv=5).fit(X_train, y_train)
```

- 의사결정나무분석의 초모수인 나무의 최대깊이와 불순도측도를 각각 설정하였음. 나무의 최대깊이는 1~7까지 단계적인 값과 최대깊이를 설정하지 않은 경우를 설정하였고, 불순도 측도는 gini 지수와 entropy 지수 두가지를 각각 사용하였음. 그리고 cv옵션에 5를 입력하여 5분할 형태로 자료를 분할하였음.

```
In [9]: print(GS1_1.best_score_)
0.9665780141843973

In [10]: print(GS1_1.best_params_)
{'criterion': 'entropy', 'max_depth': 4}
```

- best_score_ 함수와 best_params_ 함수를 통해 최적의 초모수와 정분류률을 확인한 결과 entropy지수와 최대깊이를 4로 하였을 때 정분류률이 0.966578014로 가장 높게 나타나는 것을 확인할 수 있음.

```
In [11]: clf1 = GS1_1.best_estimator_.fit(X_train, y_train)

In [12]: print(clf1.score(X_train,y_train))
0.99581589958159

In [13]: print(clf1.score(X_test, y_test))
0.9514563106796117
```

- 이러한 초모수 값을 이용하여 학습데이터의 정분류률을 score() 함수를 통해 계산하였더니 0.9958158995로 나타났고, 시험데이터의 정분류률은

0.95145631로 나타났음.

- 학습데이터의 정분류률와 시험데이터의 정분류률이 비슷한 값을 나타나는 것을 통해 모형이 안정화되어 있다는 것을 알 수 있음.

- 다음으로 SVM의 최적의 초모수를 결정하기 위해 GridSearchCV() 함수를 사용하였음.

```
In [14]: from sklearn.model_selection import GridSearchCV
    ...: from sklearn.svm import SVC

In [15]: param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]

In [16]: param_grid=[{'C': param_range, 'kernel': ['linear']},
    ...:             {'C': param_range, 'gamma': param_range, 'kernel': ['rbf']}]

In [17]: GS2_1 = GridSearchCV(estimator=SVC(random_state=1),
    ...:                      param_grid=param_grid,
    ...:                      scoring='accuracy', cv=5).fit(X_train, y_train)
```

- SVM의 초모수인 C, gamma, kernel을 각각 설정하였음. C와 gamma는 0.0001에서 1000까지 한 단위씩 증가하도록 를 설정하였고, kernel은 linear와 rbf 두 가지를 각각 사용하였음. 그리고 cv옵션에 5를 입력하여 5분할 형태로 자료를 분할하였음.

```
In [18]: print(GS2_1.best_score_)
0.9958333333333332

In [19]: print(GS2_1.best_params_)
{'C': 1.0, 'kernel': 'linear'}
```

- best_score_ 함수와 best_params_ 함수를 통해 최적의 초모수와 정분류률을 확인한 결과 C가 1, kernel을 linear로 하였을 때 정분류률이 0.99583333으로 가장 높게 나타나는 것을 확인할 수 있음.

```
In [20]: clf2 = GS2_1.best_estimator_.fit(X_train, y_train)

In [21]: print(clf2.score(X_train,y_train))
0.9916317991631799

In [22]: print(clf2.score(X_test, y_test))
0.9805825242718447
```

- 이러한 초모수 값을 이용하여 학습데이터의 정분류률을 score() 함수를 통해 계산하였더니 0.91631799로 나타났고, 시험데이터의 정분류률은 0.98058252로 나타났음.
- 학습데이터의 정분류률와 시험데이터의 정분류률이 비슷한 값을 나타나는 것을 통해 모형이 안정화되어 있다는 것을 알 수 있음.

- 또한, SVM의 경우 의사결정나무과 정분류률이 비슷하지만 시험데이터의 정분류률이 의사결정나무분석에 비해 안정화되어 있어 의사결정나무분석에 비해 SVM이 더 효과적인 분석모델이라는 것을 알 수 있음.

- 다음으로 중첩 교차검증 방법을 통해 의사결정나무분석의 최적의 초모수를 결정하고 모형을 평가해보도록 하겠음.
- 먼저 자료를 무작위 층화 추출을 하기 위해 StratifiedKFold() 함수를 사용하였음.

```
In [27]: X=df[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']]

In [28]: y=df['species']

In [29]: from sklearn.model_selection import StratifiedKFold

In [30]: inner_cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=1)

In [31]: outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
```

- 중첩 교차검증 방법은 전체 데이터를 사용하여 Outer loop와 Inner loop를 설정하므로 전체 데이터를 각각 입력변수 X와 출력변수 y에 할당하였음.
- 그리고 inner loop는 3회, outer loop는 5회 분할하기 위해 n_splits 옵션에 각각 3과 5를 입력하였음. 그리고 무작위 추출을 위해 shuffle 옵션에 True를 입력하였음.

- 다음으로 inner loop를 통해 의사결정나무분석의 최적의 초모수를 결정하기 위해 GridSearchCV() 함수를 사용하였음.

```
In [32]: from sklearn.model_selection import GridSearchCV
    ...: from sklearn.tree import DecisionTreeClassifier

In [33]: GS1_2 = GridSearchCV(estimator=DecisionTreeClassifier(random_state=1),
    ...:                      param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'criterion':['gini']},
    ...:                                  {'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'criterion': ['entropy']}],
    ...:                      scoring='accuracy', cv=inner_cv)
```

- 의사결정나무분석의 초모수인 나무의 최대깊이와 불순도측도를 각각 설정하였음. 나무의 최대깊이는 1~7까지 단계적인 값과 최대깊이를 설정하지 않은 경우를 설정하였고, 불순도 측도는 gini 지수와 entropy 지수 두가지를 각각 사용하였음. 그리고 cv옵션에 StratifiedKFold() 함수를 통해 생성한 inner_cv 객체를 입력하였음.

- 다음으로 outer loop를 통해 의사결정나무분석의 정분류률을 계산하기 위해 cross_val_score() 함수를 사용하였음.

```
In [34]: from sklearn.model_selection import cross_val_score,cross_validate

In [35]: scores = cross_val_score(GS1_2, X, y, scoring='accuracy', cv=outer_cv)

In [36]: print(scores)
[0.97101449 0.98550725 0.98529412 0.92647059 0.91176471]

In [37]: from numpy import mean, std

In [38]: print('CV accuracy: %.3f +/- %.3f' % (mean(scores), std(scores)))
CV accuracy: 0.956 +/- 0.031
```

- 5회에 걸친 outer loop를 통해 나타난 정분류률이 각각 0.97101, 0.98550, 0.98529, 0.92647, 0.91176으로 나타났음. 이를 평균과 분산을 구하기 위해 numpy 모듈의 mean과 std 함수를 사용하여 계산한 결과 중첩 교차검증을 통해 나타난 의사결정나무분석의 정분류률은 평균 0.956, 표준편차 0.031이 나타났음.

- 다음으로 inner loop를 통해 SVM의 최적의 초모수를 결정하기 위해 GridSearchCV() 함수를 사용하였음.

```
In [39]: from sklearn.model_selection import GridSearchCV
    ...: from sklearn.svm import SVC

In [40]: param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]

In [41]: param_grid=[{'C': param_range, 'kernel': ['linear']},
    ...:             {'C': param_range, 'gamma': param_range, 'kernel': ['rbf']}]

In [42]: GS2_2 = GridSearchCV(estimator=SVC(random_state=1),
    ...:                      param_grid=param_grid,
    ...:                      scoring='accuracy', cv=inner_cv)
```

- SVM의 초모수인 C, gamma, kernel을 각각 설정하였음. C와 gamma는 0.0001에서 1000까지 한 단위씩 증가하도록 를 설정하였고, kernel은 linear와 rbf 두 가지를 각각 사용하였음. 그리고 cv옵션에 StratifiedKFold() 함수를 통해 생성한 inner_cv 객체를 입력하였음.

- 다음으로 outer loop를 통해 SVM의 정분류률을 계산하기 위해 cross_val_score() 함수를 사용하였음.

```
In [43]: from sklearn.model_selection import cross_val_score,cross_validate

In [44]: scores = cross_val_score(GS2_2, X, y, scoring='accuracy', cv=outer_cv)

In [45]: print(scores)
[0.97101449 0.98550725 1.         1.         0.97058824]

In [46]: from numpy import mean, std

In [47]: print('CV accuracy: %.3f +/- %.3f' % (mean(scores), std(scores)))
CV accuracy: 0.985 +/- 0.013
```

- 5회에 걸친 outer loop를 통해 나타난 정분류률이 각각 0.97101, 0.98550, 1.000 1.000, 0.970588로 나타났음. 이를 평균과 분산을 구하기 위해 numpy 모듈의 mean과 std 함수를 사용하여 계산한 결과 중첩 교차검증을 통해 나타난 SVM의 정분류률은 평균 0.985, 표준편차 0.013이 나타났음.
- 중첩 교차검증을 통해 SVM의 경우 의사결정나무에 비해 정분류률이 높은 것으로 보아 의사결정나무분석에 비해 SVM이 더 효과적인 분석모델이라는 것을 알 수 있음.