교차 검증(Cross Validation)

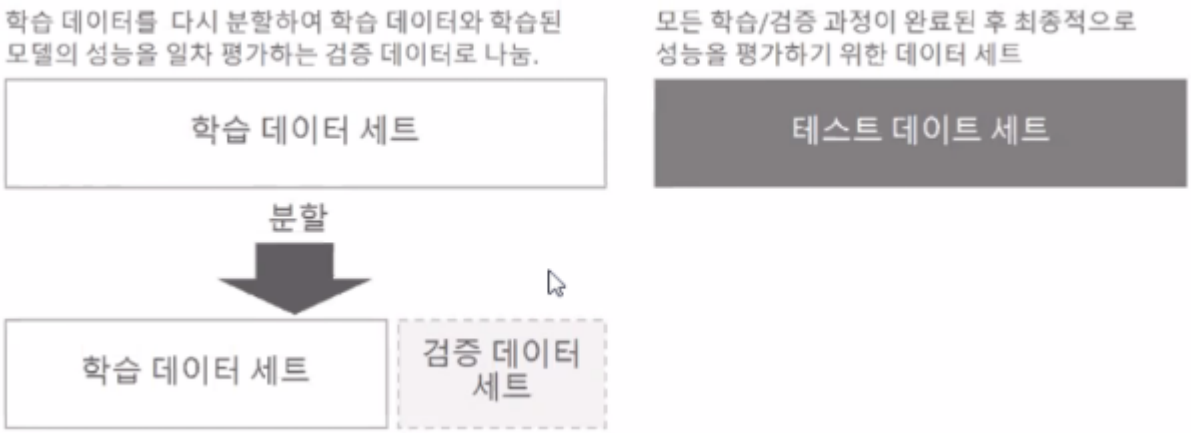
교차 검증이 필요한 이유

- 사용할 데이터가 있을 때 그 데이터는 label이 있는 train, test set으로 구성되어 있음
    - 만약 'train set을 다시 train set + validation set으로 분리하지 않는다'라고 가정하면, 모델 검증을 위해서 test set을 사용하여야 할 것임
        - 사실상 test set이 아닌 valdiation set인 셈인데, 여기에 한 가지 약점이 존재함
        - 고정된 test set을 가지고 모델의 성능을 확인하고 파라미터를 수정하고, 이 과정을 반복하면 결국 내가 만든 모델은 test set에만 잘 동작하는 모델이 됨
- 이 경우에는 test set에 과적합(overfitting)되어 다른 실제 데이터를 가지고 예측을 수행하면 엉망인 결과가 나와버리게 됨

교차검증이란?

- 훈련 데이터 세트를 바꿔가며 훈련하면서 나온 평균을 정확도로 보는 방법을 뜻함
    - 고정된 train set과 test set으로 평가를 하고, 반복적으로 모델을 튜닝하다보면 test set에만 과적합되어 버리는 결과가 생기는 문제를 해결

- 훈련 때 학습 데이터에 과도하게 초점을 맞춰 머신이 훈련될 수가 있는데 이 같은 경우에는 훈련시에는 점수가 잘 나오지만 실제 테스트를 할 때는 점수가 잘 나오지 않음 -> 과적합 (overfitting) 원인



교차 검증의 장점과 단점

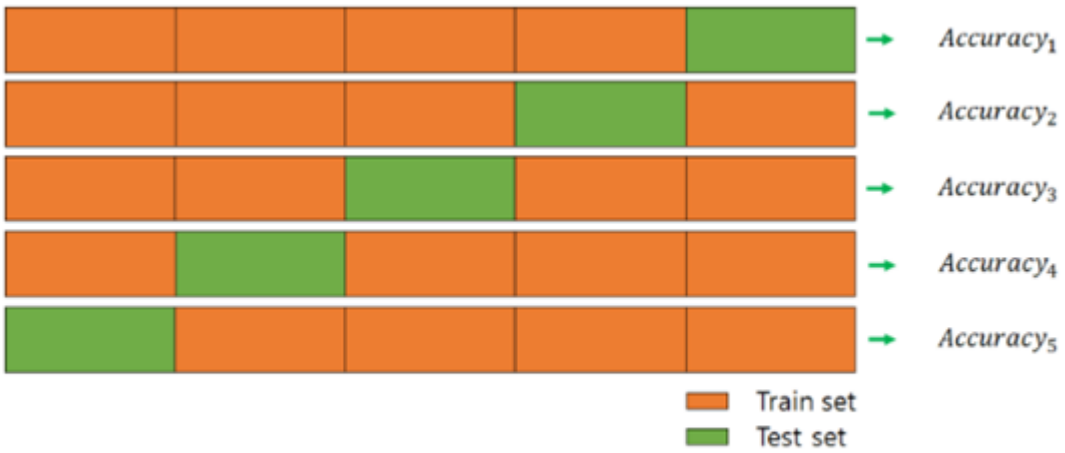
- 장점
  - 모든 데이터 셋을 평가에 활용할 수 있다.
    - 평가에 사용되는 데이터 편중을 막을 수 있다.
    - 특정 평가 데이터 셋에 overfit 되는 것을 방지할 수 있다.
    - 평가 결과에 따라 좀 더 일반화된 모델을 만들 수 있다.
  - 모든 데이터 셋을 훈련에 활용할 수 있다.
    - 정확도를 향상시킬 수 있다.

        - 데이터 부족으로 인한 underfitting을 방지할 수 있다.
- 단점
    - Iteration 횟수가 많기 때문에 모델 훈련/평가 시간이 오래 걸린다.

---

k-겹 교차 검증(k-fold cross validation)

- 데이터를 k개의 데이터 폴드로 분할하고, 각 Iteration마다 test set을 다르게 할당하여 총 k개의 '데이터 폴드 세트(Figure 4의 경우 train fold 4개 + test fold 1개)'를 구성
    - 따라서 모델을 학습 및 훈련하는데 총 k번의 Iteration이 필요한 것이 특징임
    - 각 데이터 폴드 세트에 대해서 나온 검증 결과들을 평균내어 최종적인 검증 결과를 도출하는 것이 일반적임



$$Accuracy = Average(Accuracy_1, \cdots, Accuracy_k)$$

```
#https://continuous-development.tistory.com/entry/MLDL-python-%EC%9D%84-%ED%86%B5%ED%95%9C-%EA%B5%90%EC%B0%A8%EA%B2%80%EC%A6%9D-k-Fold-strati
#sklearn을 통해 KFold를 import
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
import pandas as pd
import numpy  as np
```

```
fold_iris.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
# sklearn을 통해 데이터 셋을 가져옴
# data(독립변수)와 target(종속변수)를 가져옴
# 여기서는 DecisionTree를 사용
fold_iris = load_iris()
features = fold_iris.data
# print(features)
label    = fold_iris.target
# print(label)
fold_df_clf = DecisionTreeClassifier()
```

```
# 5개의 폴드 세트를 분리하여 각 폴드 세트별 정확도를 담을 리스트를 생성
kfold = KFold(n_splits=5)
cv_accuracy = []
```

```
#KFold를 5개로 split, 값은 사용자가 임의로 정할 수 있음
n_iter = 0
for train_idx , test_idx in kfold.split(features) :
    X_train , X_test = features[train_idx] , features[test_idx]

```

```
    y_train , y_test = label[train_idx] , label[test_idx]

    # 학습을 진행하겠다면?
    fold_df_clf.fit(X_train , y_train)
    # 예측
    fold_pred = fold_df_clf.predict(X_test)

    # 정확도 측정
    n_iter += 1
    accuracy = np.round(accuracy_score(y_test , fold_pred), 4)
    print('₩n{} 교차검증 정확도 : {} , 학습 데이터 크기 : {} , 검증 데이터 크기 : {}'.format(n_iter, accuracy, X_train.shape[0] , X_test.shap

    cv_accuracy.append(accuracy)
print('₩n')

print('₩n 평균검증 정확도 :  ' , np.mean(cv_accuracy))
```

```
 1 교차검증 정확도 : 1.0 , 학습 데이터 크기 : 120 , 검증 데이터 크기 : 30

 2 교차검증 정확도 : 1.0 , 학습 데이터 크기 : 120 , 검증 데이터 크기 : 30

 3 교차검증 정확도 : 0.8333 , 학습 데이터 크기 : 120 , 검증 데이터 크기 : 30

 4 교차검증 정확도 : 0.9333 , 학습 데이터 크기 : 120 , 검증 데이터 크기 : 30

 5 교차검증 정확도 : 0.8333 , 학습 데이터 크기 : 120 , 검증 데이터 크기 : 30


 평균검증 정확도 :  0.91998
```

```
#https://www.analyticsvidhya.com/blog/2021/11/top-cross-validation-techniques-with-python-code/
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
iris=load_iris()
X=iris.data
Y=iris.target
print("Size of Dataset {}".format(len(X)))
logreg=LogisticRegression()
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=42)
logreg.fit(x_train,y_train)
predict=logreg.predict(x_test)
print("Accuracy score on training set is {}".format(accuracy_score(logreg.predict(x_train),y_train)))
print("Accuracy score on test set is {}".format(accuracy_score(predict,y_test)))
```

```
Size of Dataset 150
Accuracy score on training set is 0.9619047619047619
Accuracy score on test set is 1.0
```

홀드아웃 방법(Holdout method)

- 주어진 train set을 다시 임의의 비율로 train set과 test set으로 분할하여 사용

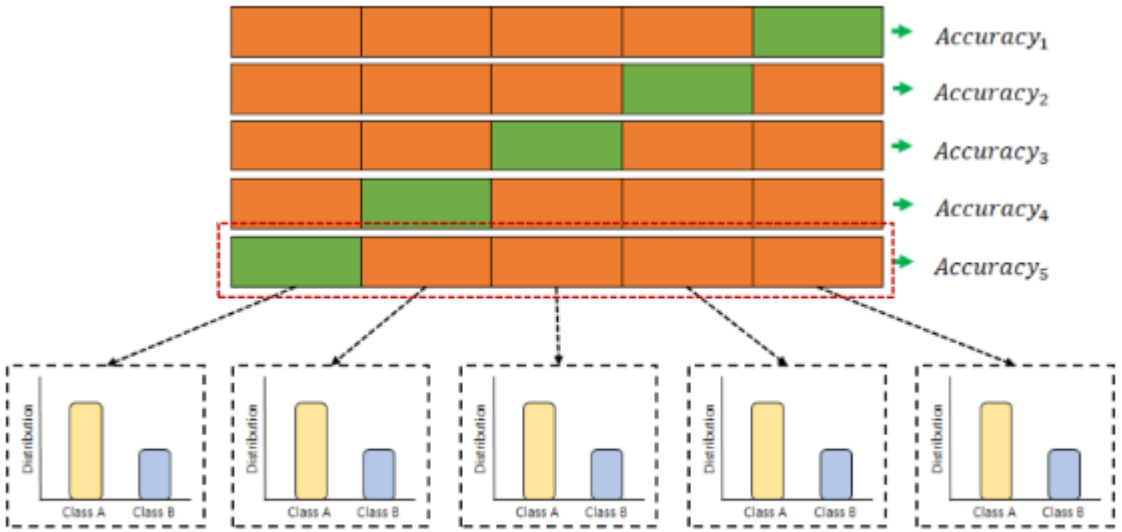
  - 이때 train : test = 9 : 1 또는 7 : 3 비율이 가장 자주 쓰임
  - 장점 - Iteration(훈련 및 검증)을 한 번만 하기 때문에 계산 시간에 대한 부담이 적음
  - 단점 - test set에 관한 검증 결과 확인 후 모델 파라미터 튜닝을 하는 작업을 반복하게 되면 모델이 test set에 대해 overfit될 가능성이 높음

계층별 k-겹 교차 검증(Stratified k-fold cross validation)

- 주로 Classification 문제에서 사용되며, label의 분포가 각 클래스별로 불균형을 이룰 때 유용하게 사용

  - label의 분포가 불균형한 상황에서 sample의 index 순으로 데이터 폴드 세트를 구성하는 것은 데이터를 검증하는데 치명적인 오류를 야기할 수 있음
  - Stratified k-fold cross validation은 이러한 데이터 label의 분포까지 고려해 주어서(그렇기 때문에 데이터 폴드 세트를 구성해주는 함수에서 데이터의 label 값이 요구됨), 각 훈련 또는 검증 폴드의 분포가 전체 데이터셋이 가지고 있는 분포에 근사하게 됨

- target에 속성값의 개수를 동일하게 하게 가져감으로써 kfold 같이 데이터가 한곳으로 몰리는것을 방지함



$$Accuracy = Average(Accuracy_1, \cdots, Accuracy_k)$$

```
# [실습] random_state = 100
# 붓꽃 데이터 세트에서 Stratified KFold 를 이용하여 교차검증(3, 5)을 진행하고 평균정확도를 확인
# Stratified KFold 는 분류 , 회귀 X
# 회귀는 연속된 숫자 값이기 때문에 지원하지 않음.


from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold

# 회귀에서는 지원하지 않음
from sklearn.model_selection import StratifiedKFold

import pandas as pd
import numpy  as np

result_iris = load_iris()
result_features = result_iris.data
```

```
result_label    = result_iris.target

# 학습기 생성
result_clf = DecisionTreeClassifier(random_state=100)
```

```
result_skfold = StratifiedKFold(n_splits=3)
idx_iter=0
cv_accuracy=[]

# StratifiedKFold의 split( ) 호출시 반드시 레이블 데이터 셋도 추가 입력 필요
for train_index, test_index  in result_skfold.split(features, label):
    # split( )으로 반환된 인덱스를 이용하여 학습용, 검증용 테스트 데이터 추출
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]

    #학습 및 예측
    result_clf.fit(X_train , y_train)
    pred = result_clf.predict(X_test)

    # 반복 시 마다 정확도 측정
    idx_iter += 1
    accuracy = np.round(accuracy_score(y_test,pred), 4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]

    print('Wn#{0} 교차 검증 정확도 :{1}, 학습 데이터 크기: {2}, 검증 데이터 크기: {3}'
          .format(idx_iter, accuracy, train_size, test_size))
    print('#{0} 검증 세트 인덱스:{1}'.format(idx_iter,test_index))
    cv_accuracy.append(accuracy)
```

```
#1 교차 검증 정확도 :0.98, 학습 데이터 크기: 100, 검증 데이터 크기: 50
#1 검증 세트 인덱스:[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  50
  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 100 101
 102 103 104 105 106 107 108 109 110 111 112 113 114 115]
```

```
#2 교차 검증 정확도 :0.92, 학습 데이터 크기: 100, 검증 데이터 크기: 50
#2 검증 세트 인덱스:[ 17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  67
  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82 116 117 118
 119 120 121 122 123 124 125 126 127 128 129 130 131 132]

#3 교차 검증 정확도 :0.96, 학습 데이터 크기: 100, 검증 데이터 크기: 50
#3 검증 세트 인덱스:[ 34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  83  84
  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 133 134 135
 136 137 138 139 140 141 142 143 144 145 146 147 148 149]
```

```
# 교차 검증별 정확도 및 평균 정확도 계산
print('Wn## 교차 검증별 정확도:', np.round(cv_accuracy, 4))
print('## 평균 검증 정확도:', np.mean(cv_accuracy))
```

```
## 교차 검증별 정확도: [0.98 0.92 0.96]
## 평균 검증 정확도: 0.9533333333333333
```

<<<참조자료 사이트>>>

1.교차 검증이 필요한 이유

2.Top 7 Cross-Validation Techniques with Python Code

3.python을 통한 교차검증 (k -Fold , stratifiedkFold)

1.Easy! 딥러닝 4-2강. K-fold Cross Validation (교차 검증) 진짜 쉽게 설명해 드려요

2.K-겹 교차 검증 (K-fold cross validation)
