교차검증

1. Test vs Training

- Model이 Sample에 얼마나 일반화 될지 아는 방법은 무엇일까?
  - 실제 적용해보면 된다.
  - 모델을 만들고 적용해보면 성능이 나옴
  - 다른방법은?
- Traning set과 Test set으로 분리
  - 훈련 세트와 테스트 세트 두개로 분리
  - 훈련 세트를 사용해 모델을 훈련하고 테스트 세트를 사용해 모델을 테스트
  - 보통 데이터(1만개) 80%는 Training Set, 20%는 Test Set 으로 떼어 놓는다.
  - 10만개 정도면 99% 는 Training Set, 1% 는 Test Set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(DATA이름, DATA target)
```

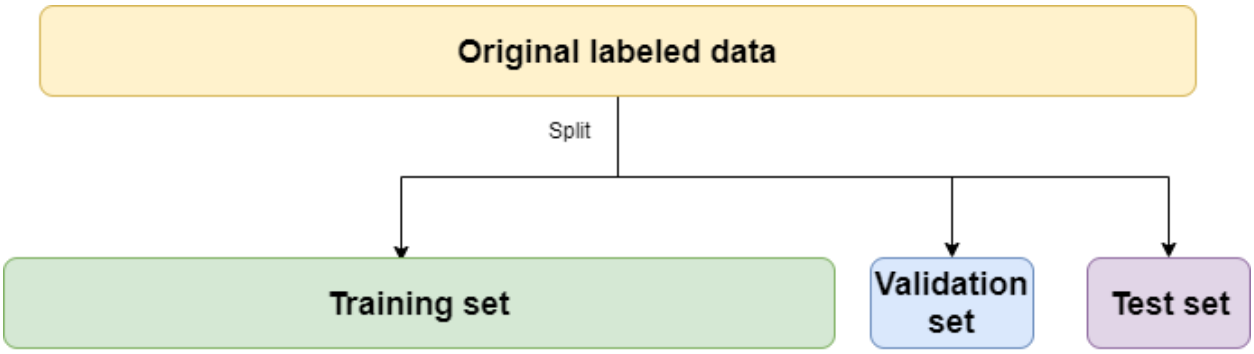
- random_state : Random seed 번호, 번호를 지정하면 동일한 데이터 세트로 분리해 준다.
- test_size : 전체 데이터에서 테스트 데이터 세트 크기를 얼마로 샘플링 할것인가(기본 25%)
- train_size : test_size의 남은 값
- shuffle : 데이터를 분리하기 전에 섞을지 결정

2. Validation Set의 필요성

- 우리가 사용한 데이터는 Train + Test 로 구성되어 있다.
- 이런경우 모델을 검증하기 위해 Test Set을 사용하는데, 사실상 Test Set = Validation Set이 된다.
- 이 경우 문제점이 무엇일까?
    - Test Set을 이용해 모델의 성능을 확인하고 파라미터를 수정하였으므로, Test Set에서만 잘 동작하는 모델이 완성된다.

2.1 Validtion Set

- 데이터 세트를 학습 세트(Training Set), 검증 세트(Validation Set), 테스트 세트(Test Set)로 나누어 테스트 세트에 과적합한 모델이 될 가능성을 크게 낮출수 있다.
-



- 검증 세트를 통해 모델을 선정하는 과정
    1. Training Set으로 Model 학습
    2. 학습된 Model을 Validation Set으로 평가
    3. Validation Set 로 가한 결과에 따라 모델을 조정하고 다시 학습
    4. 가장 우수한 결과를 보이는 모델을 선택
    5. 그 모델을 이용해 Test Set으로 평가

~~6. 그 모델을 이용해 Test Set으로 평가~~

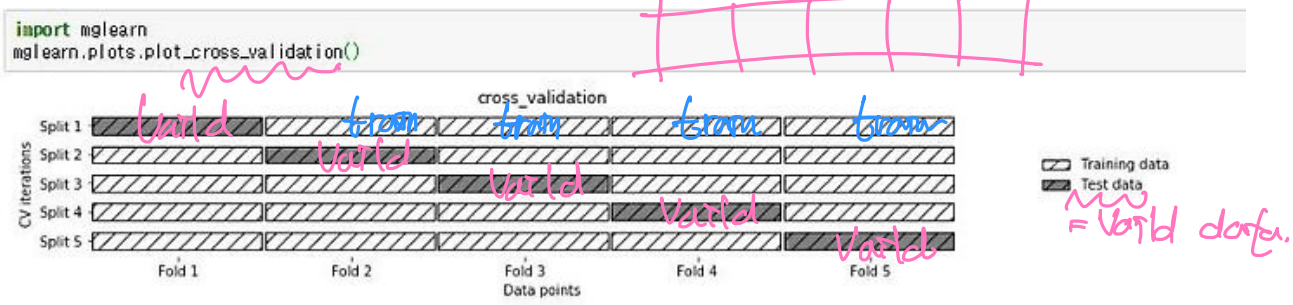
- Test Set과 달리 model 생성시 Model의 성능을 평가한다면 어떤점이 좋을까?
  - Overfitting 을 방지할수 있다.
  - Validation Set의 결과와 Training Set의 결과의 차이가 벌어지면 Overfitting 이므로!
  - 결국 최적의 Parameter를 찾음
- Training 중간중간 Model의 성능을 확인 할수 있다.
- 과정
  - Model1 → 평가 (epoch1)
  - Model2 → 평가 (epoch2)
  - ...
- 보통 6:2:2 또는 8:1:1 로 나눈다.

Model의 성능측정

- Training Set
  - 모델 생성시 사용
- Validation Set
  - 모델의 성능 확인
  - 모델이 제대로 만들어 지고 있나?
- Test Set
  - 모델 평가
  - 실제로 사용하기 전에 평가

교차 검증종류

1. 단순 교차검증 cross_val_score



- k=5일 때, 즉 데이터를 5개의 부분 집합으로 분할한 후, 각 분할마다 하나의 폴드를 테스트용으로 사용하고 나머지 4개의 폴드는 훈련용으로 사용.
- 이 과정을 반복하여 각 분할마다 정확도를 측정한다.
- 사이킷런에서는 교차검증을 위해 cross_val-score 함수를 제공한다.

```
from sklearn.datasets import load_iris
iris = load_iris()
```

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

```
logreg = LogisticRegression()
```

```
score = cross_val_score(logreg, iris.data, iris.target, cv=5)
print('cross validation score : %s'% score)
```

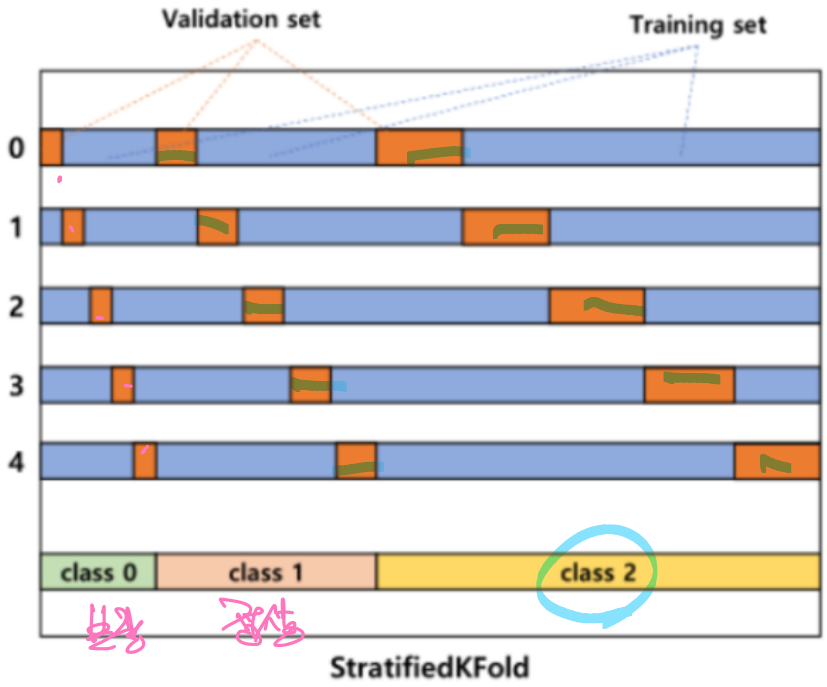
```
cross validation score : [1.         0.96666667 0.93333333 0.9        1.        ]
```

- **cross_val_score(모델 명, 훈련데이터, 타깃, cv)인데, 여기에서 cv는 폴드(fold)수**
- 기본 default는 3이고, 그림처럼 cv=5를 해주면 '5-겹 교차검증'을 하라는 의미가 된다.
- 최종적으로 평균을 내어 정확도를 간단히 한다.

```
print('교차검증 평균 : %.3f' %(score.mean()))
```

```
교차검증 평균 : 0.960
```

2. 계층별 k-겹 교차검증



- 데이터가 편향되어 있을 경우(몰려있을 경우) 단순 k-겹 교차검증을 사용하면 성능 평가가 잘 되지 않을 수 있다.
- 따라서 이럴 땐 **stratified k-fold cross-validation**을 사용한다.
- 계층을 나누고 분할해서 뽑는다.

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=0)

score = cross_val_score(logreg, iris.data, iris.target, cv=skf)

print(score.mean())
```

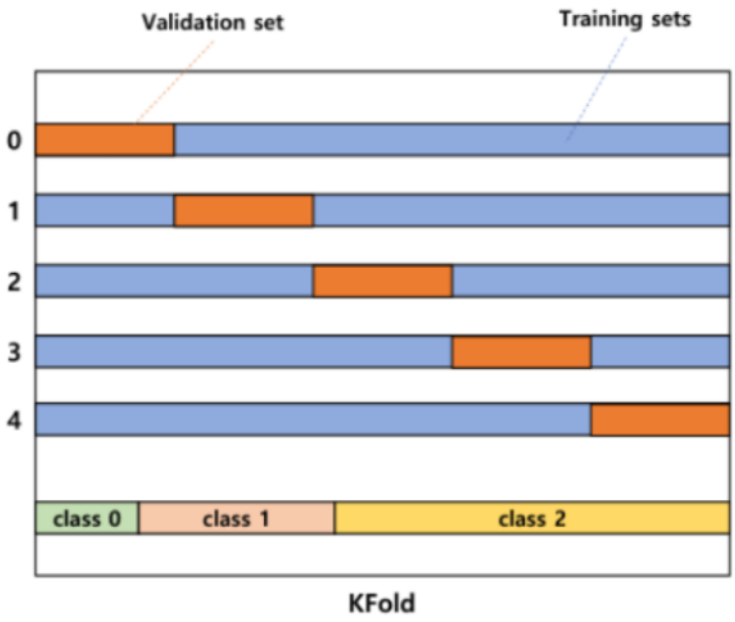
```
0.96
```

- StratifiedKFold(n_splits, shuffle, random_state)
- **n_splits**은 몇 개로 분할할지를 정하는 매개변수
- **shuffle의 기본값 False 대신 True**를 넣으면 Fold를 나누기 전에 무작위로 섞는다.
- **cross_val_score함수의 cv 매개변수에 넣으면 된다**.

참고

- 일반적으로 회귀에는 기본 k-겹 교차검증을 사용하고, 분류에는 StratifiedKFold를 사용
- 또한, cross_val_score 함수에는 KFold의 매개변수를 제어할 수가 없으므로, 따로 KFold 객체를 만들고 매개변수를 조정한 다음에 cross_val_score의 cv 매개변수에 넣어야 한다.

3. KFold 상세조정



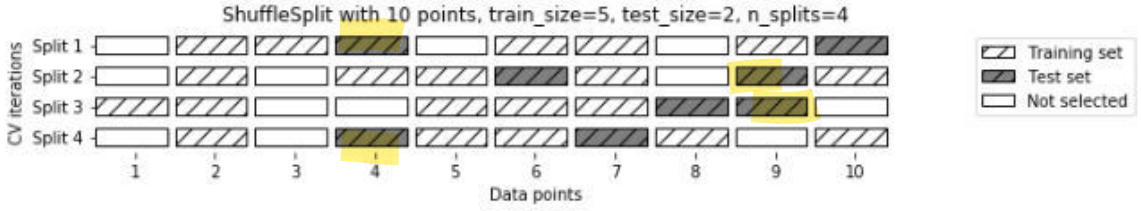
- cross_val_score함수에서는 cv로 폴드의 수를 조정할 수 있었다.
- 반면, 위에서 StratifiedKFold의 매개변수를 조정하고 이를 skf에 넣은 후, cross_val_score의 cv에 전달했었다.
- 이렇게 cv 매개변수로 전달되는 것을 '**교차 검증 분할기**'라고 하는데, 이를 통해 데이터 분할을 좀 더 세밀하게 할 수 있다.

```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=6, shuffle=True, random_state=0)
score = cross_val_score(logreg, iris.data, iris.target, cv=kfold)
print('cross validation score : {}'.format(score))
print('mean score : {:.2f}'.format(score.mean()))
```

```
cross validation score : [0.96 0.92 0.92 0.96 1.   0.88]
mean score : 0.94
```

4. 임의분할 교차검증(shuffle split cross validation)

```
mglearn.plots.plot_shuffle_split()
```



- 임의분할 교차검증은 train set과 test set의 크기를 유연하게 조절해야 할 때 유용
- train_size와 test_size에 정수를 입력하면 해당 수 만큼 데이터포인트의 개수가 정해지며, 만일 실수를 입력하면 비율이 정해진다.

```
from sklearn.model_selection import ShuffleSplit
shuffle_split = ShuffleSplit(train_size=0.5, test_size=0.5, random_state=0, n_splits=8)
score = cross_val_score(logreg, iris.data, iris.target, cv=shuffle_split)
print('Shuffle Split score : ₩n{}'.format(score))
```

```
Shuffle Split score :
[0.84        0.93333333 0.90666667 1.         0.90666667 0.93333333
 0.94666667 1.        ]
```

- train_size를 전체데이터의 50%, test_size를 전체데이터의 50%로 설정하고, 8번 반복 분할하고 실행 때마다 같은 결과가 나오도록 했다.