T-Academy

Lab_05) Model Validation

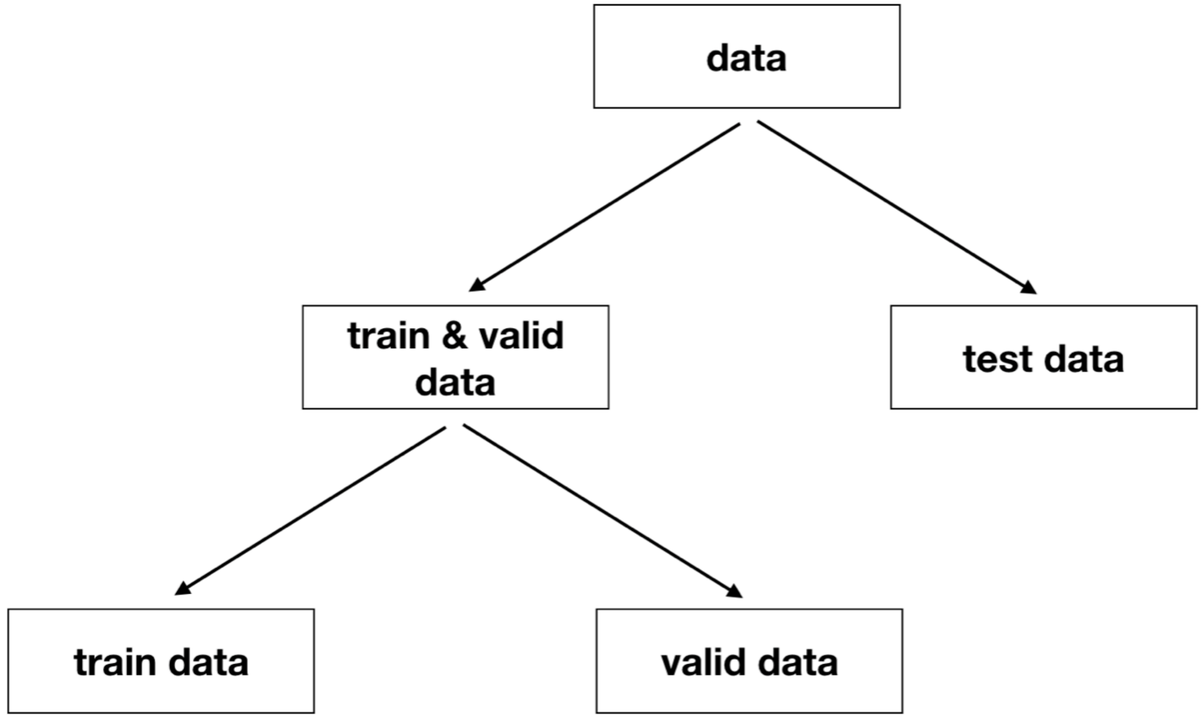
2019. Apr. 25.
SK플래닛 T아카데미
캐글 코리아
강천성

Cross Validation

1. Train, Valid, Test Set

훈련, 검증, 테스트 데이터라고 부르는 3가지를 한번 이야기 해보겠습니다.

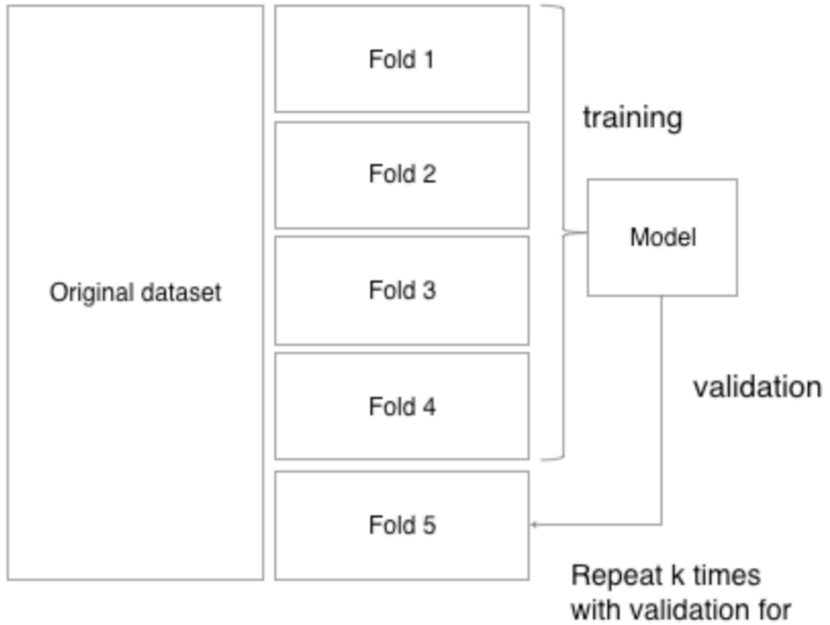
- Train Data : 모델을 학습하는데 사용하는 데이터 (모델이 알고 있는 학습할 데이터)
- Valid Data : 학습한 모델의 성능을 검증하는 데이터 (모델이 모르는 학습하지 않을 데이터, 모델 검증에 사용하는 데이터)
- Test Data : 학습한 모델로 예측할 데이터 (모델이 모르는 예측할 데이터)



머신러닝에서 Validation 데이터가 왜 필요한지에 대한 부분은 참조 링크를 남겨두었으니 확인하시면 좋겠습니다.

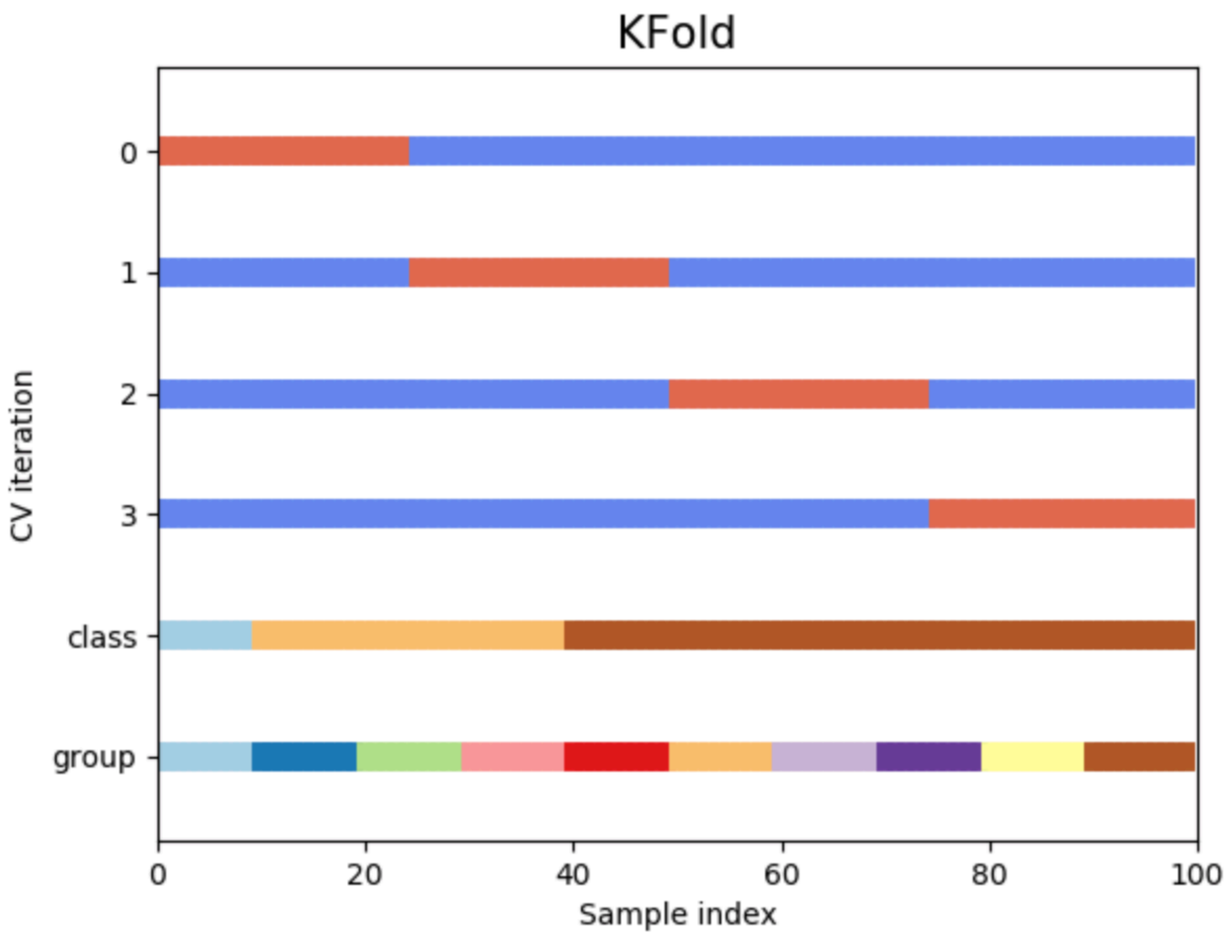
2. k-fold with stratify

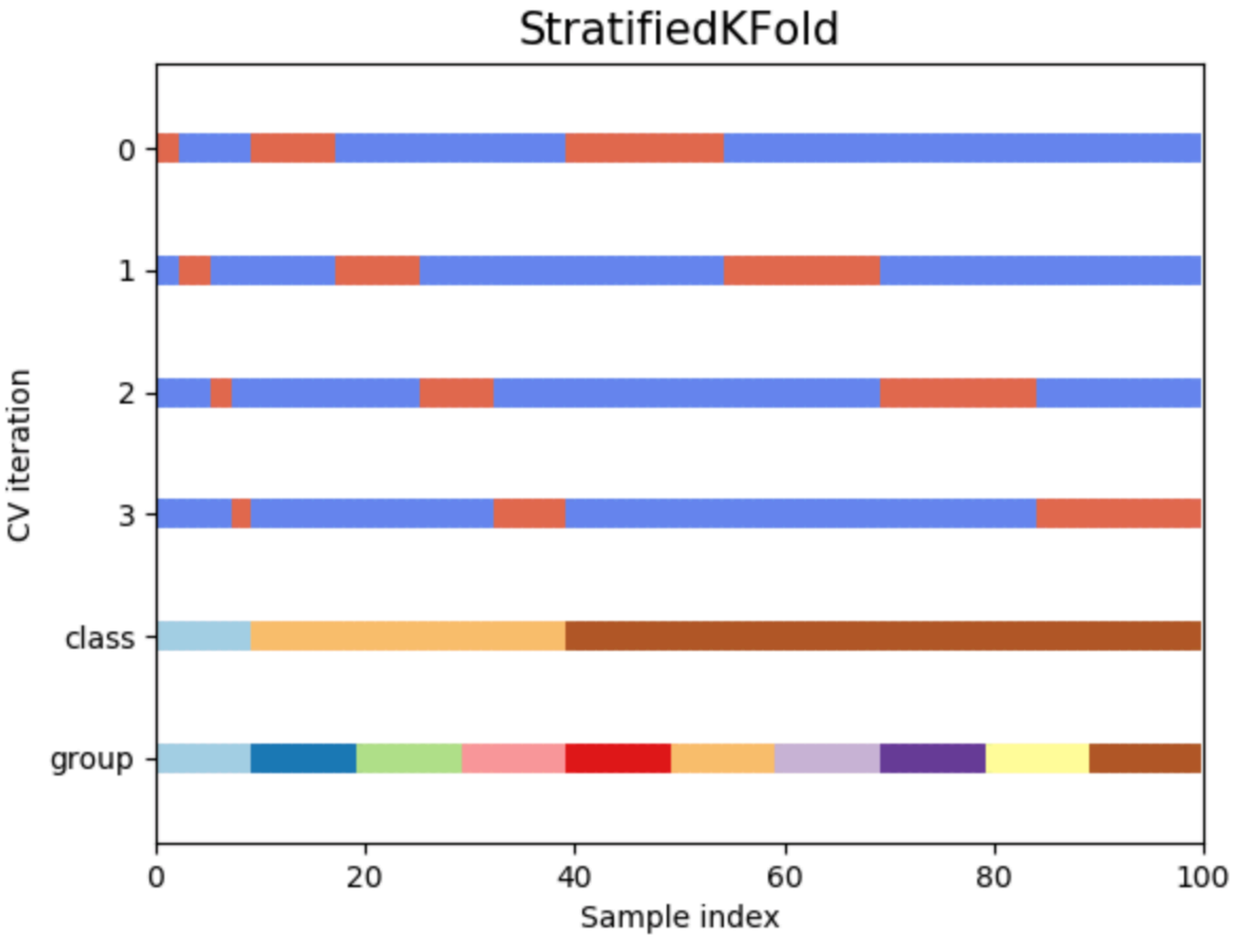
k-fold는 데이터를 k개로 쪼개는 것을 말합니다.
일반적으로 Cross Validation에서 사용되며, 데이터셋을 k개로 쪼개어 k-1개로 모델을 학습하고, 1개로 모델을 검증합니다.
k개로 데이터를 쪼개면, 모든 fold에 대해(하나의 fold를 선택하여) 검증하는 방식으로 k번 다른 데이터셋으로 학습한 모델을 검증할 수 있습니다.



Stratify, 계층적 k-fold는 뭔가요?

k-fold는 데이터의 정렬 유무와 분류할 클래스의 비율에 상관없이 순서대로 데이터를 분할하는 특징이 있습니다.
하지만, 분류할 클래스의 비율이 다르다면 어떻게 될까요? 그런 경우에는, 각 fold가 학습 데이터셋을 대표한다고 말하기 어려워집니다.
한 fold에 특정 클래스가 많이 나올수도, 적게 나올수도 있기 때문입니다. Stratified k-fold는 그러한 문제점을 해결하기 위해 제안되었습니다.
k개의 fold도 분할한 이후에도, 전체 훈련 데이터의 클래스 비율과 각 fold가 가지고 있는 클래스의 비율을 맞추어 준다는 점이 기존의 k-fold와의 다른 특징 입니다.


KFold
CV iteration
0
1
2
3
class
group
0
20
40
60
80
100
Sample index


StratifiedKFold
CV iteration
0
1
2
3
class
group
0
20
40
60
80
100
Sample index

• k-Fold

1. 모델 불러오기 및 정의

```python
from sklearn.model_selection import KFold
kf = KFold(n_splits=5, random_state=2019)
```

2. split by fold

```python
for i, (trn_idx, val_idx) in enumerate(kf.split(kf_data.values, kf_label)) :
    trn_data, trn_label = kf_data.values[trn_idx,:], kf_label[trn_idx]
    val_data, val_label = kf_data.values[val_idx,:], kf_label[val_idx]

    print('{} Fold, trn label\n {}'.format(i, trn_label))
    print('{} Fold, val label\n {}\n'.format(i, val_label))
```

3. 결과 확인

```
0 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
0 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

1 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
1 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1]

2 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
2 Fold, val label
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
3 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
3 Fold, val label
 [1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

4 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
4 Fold, val label
 [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

• Stratified k-Fold

1. 모델 불러오기 및 정의 ·····················→

```python
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, random_state=2019)
```

2. split by fold ·····················→

```python
for i, (trn_idx, val_idx) in enumerate(skf.split(kf_data.values, kf_label)) :
    trn_data, trn_label = kf_data.values[trn_idx,:], kf_label[trn_idx]
    val_data, val_label = kf_data.values[val_idx,:], kf_label[val_idx]

    print('{} Fold, trn label\n {}'.format(i, trn_label))
    print('{} Fold, val label\n {}\n'.format(i, val_label))
```

3. 결과 확인

```
0 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
0 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]

1 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
1 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]

2 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
2 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]

3 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
3 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]

4 Fold, trn label
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
4 Fold, val label
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]
```

Cross Validation 해보기

Stratified k-fold를 이용해 Cross Validation을 진행해 보겠습니다.

```
from sklearn.ensemble import RandomForestClassifier

val_scores = list()

for i, (trn_idx, val_idx) in enumerate(skf.split(kf_data, kf_label)) :
    trn_data, trn_label = kf_data.values[trn_idx,:], kf_label[trn_idx]
    val_data, val_label = kf_data.values[val_idx,:], kf_label[val_idx]

    # 모델 정의
    clf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=2019)

    # 모델 학습
    clf.fit(trn_data, trn_label)
    trn_acc = clf.score(trn_data, trn_label)*100
    val_acc = clf.score(val_data, val_label)*100
    print('{} Fold, train Accuracy : {:.2f}%, validation Accuracy : {:.2f}%'.format(i, trn_acc, val_acc))

    val_scores.append(val_acc)

# Mean Validation Score
print('Cross Validation Score : {:.2f}%'.format(np.mean(val_scores)))
```

```
0 Fold, train Accuracy : 100.00%, validation Accuracy : 96.67%
1 Fold, train Accuracy : 100.00%, validation Accuracy : 96.67%
2 Fold, train Accuracy : 100.00%, validation Accuracy : 93.33%
3 Fold, train Accuracy : 100.00%, validation Accuracy : 96.67%
4 Fold, train Accuracy : 100.00%, validation Accuracy : 100.00%
Cross Validation Score : 96.67%
```

Cross Validation Score

방금 전 반복문을 사용해 Cross Validation을 진행해 봤습니다.
그런데 Sklearn에는 한번에 k-fold Cross Validation Score를 계산하는 cross_val_score 함수를 제공합니다.
파라미터로 cv에 숫자를 전달하면, 그 숫자 만큼의 fold를 만들어 Cross Validation(CV)을 진행하고, kfold 객체를 전달하면 해당 객체에 맞게 데이터를 분할하여 CV Score를 계산합니다.
cross_val_score 함수는 폴드 개수대로 Score를 반환하며, 해당 스코어들의 평균을 계산해 모델의 성능을 가늠해볼 수 있습니다.

- 기본적으로 cross_val_score 함수는 입력 Label 값이 클래스로 나누어진 분류 모델인 경우 StratifiedKFold를 적용합니다.

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

```
# 숫자로 전달하는 경우
print('랜덤 포레스트 k-Fold CV Score(Acc) : {:.2f}%'.format(np.mean(cross_val_score(rf, kf_data, kf_label, cv=5))*100))
```

```
랜덤 포레스트 k-Fold CV Score(Acc) : 96.00%
```

```
# fold 객체를 전달하는 경우
print('랜덤 포레스트 k-Fold CV Score(Acc) : {:.2f}%'.format(np.mean(cross_val_score(rf, kf_data, kf_label, cv=kf))*100))
print('랜덤 포레스트 Stratify k-Fold CV Score(Acc) : {:.2f}%'.format(np.mean(cross_val_score(rf, kf_data, kf_label, cv=skf))*100))
```

```
랜덤 포레스트 k-Fold CV Score(Acc) : 75.33%
랜덤 포레스트 Stratify k-Fold CV Score(Acc) : 96.00%
```

Parameter Tuning

GridSearch

모델에는 여러가지 파라미터가 들어갑니다. SVC의 경우 Soft, Hard 마진의 정도를 결정하는 'C' 커널 함수를 결정하는 'kernel' 특정 커널에서 얼마나 세세하게 볼것인지를 결정하는 'gamma' 등
파라미터를 어떻게 결정하느냐에 따라 모델이 잘 학습하거나 잘 학습하지 못하는 경우가 발생할 수 있습니다.
Sklearn에서 가장 쉽게 제공하는 파라미터 튜닝 함수로 GridSearchCV 라는 함수가 있습니다. 해당 함수에 각 파라미터에 사용할 수치 리스트를 전달하면, 해당 함수는 파라미터들의 조합을 모두 시도해보며,
가장 좋은 성능의 파라미터를 찾게 됩니다.

간단히 GridSearchCV 함수를 사용해 랜덤 포레스트의 n_estimator, max_depth 파라미터 중 가장 좋은 파라미터 조합을 찾아보겠습니다.
GridSearchCV 함수는 Sklearn의 model_selection 패키지에 있습니다.

Scikit-Optimize

GridSearch의 단점은 사용자가 직접 파라미터에 들어갈 값들의 리스트를 지정해주어야 한다는 단점이 있습니다.
Sklearn 라이브러리 내에 존재하지는 않지만, Scikit-Optimize(이하, skopt)라는 라이브러리를 간단히 소개해 드리겠습니다.
skopt는 각 파라미터에 들어갈 값들의 최대, 최소 범위를 결정해주고 파라미터 값의 분포 스케일을 결정해주어 파라미터 튜닝을 자동화 시켜주는 라이브러리입니다.
참조 링크에 Skopt 링크 남겨드리니 확인해보시면 좋겠습니다.

• GridSearch with SVM

1. 모델 불러오기 및 정의

```
from sklearn.model_selection import GridSearchCV

params = {'n_estimators' : [50, 150, 200, 300, 400],
          'max_depth' : [2, 5 ,10, 20]}

clf = GridSearchCV(RandomForestClassifier(random_state=2019), params, cv=skf)
```

2. fit

```
clf.fit(kf_data, kf_label)
```

3. 결과 확인

```
print('GridSearchCV best score : {:.2f}%, best_params : {}'.format(clf.best_score_*100, clf.best_params_))
```

```
GridSearchCV best score : 96.67%, best_params : {'max_depth': 5, 'n_estimators': 50}
```

Ensemble

개인적으로 앙상블은 머신러닝의 꽃이라고 생각합니다. 단일 모델로 좋은 성능을 이끄는 것도 중요하지만, 서로 다른 모델의 다양성을 고려하여 결과를 이끌어내는 앙상블은 응용할 수 있는 방법이 매우 많습니다.
그 중 대표적인 3가지 앙상블에 대해 실습하고 배워보도록 하겠습니다.

1. Voting Ensemble

이름에서 알 수 있듯이 각자의 모델이 투표를 하여 클래스를 선택하는 방식의 앙상블 입니다.
Voting 앙상블은 Sklearn 자체적으로 모델로써 지원을 하며, 사용하기도 매우 쉽습니다.

다시 Adult 데이터셋으로 돌아와 앙상블을 통해 기존 단일 모델보다 좋은 결과를 얻어보도록 하겠습니다.

Voting Classifier는 Sklearn의 ensemble 패키지에 있습니다.

2. Average Blending

앙상블 기법 중 캐글에서 가장 많이 사용되는 기법입니다. Average Blending에서 회귀의 경우 각 모델들이 예측한 결과 값을 n으로 나누어 합칩니다.
분류의 경우에는 각 클래스에 해당하는 확률을 n으로 나누어 합치고, 그 중 가장 높은 확률 값을 갖는 클래스를 택하는 방식입니다.

• Voting Ensemble

1. 모델 불러오기 및 정의

```
from sklearn.ensemble import VotingClassifier
clfs = [('Logistic Regression', LogisticRegression(random_state=2019)),
        ('Random Forest', RandomForestClassifier(n_estimators=100, max_depth=10, random_state=2019)) ]

vote_clf = VotingClassifier(clfs, voting='soft')
```

2. fit

```
vote_clf.fit(x_train, y_train)
```

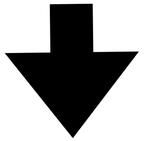
3. predict

```
y_pred = vote_clf.predict(x_test)
```

4. 결과 확인

```
print('Voting Ensemble Acc : {:.2f}%'.format(vote_clf.score(x_test, y_test)*100))
```

```
Voting Ensemble Acc : 82.51%
```



```
# 단일 모델에서 Random Forest 성능
clf = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=2019)
clf.fit(x_train, y_train)
print('Single Random Forest Acc : {:.2f}%'.format(clf.score(x_test, y_test)*100))
```

```
Single Random Forest Acc : 83.56%
```

- **Logistic Regression의 부족한 성능으로 인한 정확도 하락**
- **앙상블도 적당히 괜찮은 모델들과 섞어야 성능이 향상됨**

• Average Blending Ensemble

```
val_scores = list()

y_pred = np.zeros_like(y_test, dtype=np.float)

for i, (trn_idx, val_idx) in enumerate(skf.split(x, y)) :
    trn_data, trn_label = x.values[trn_idx, :], y.values[trn_idx]
    val_data, val_label = x.values[val_idx, :], y.values[val_idx]

    # 모델 정의
    clf = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=2019)

    # 모델 학습
    clf.fit(trn_data, trn_label)
    trn_acc = clf.score(trn_data, trn_label)*100
    val_acc = clf.score(val_data, val_label)*100
    print('{} Fold, train Accuracy : {:.2f}%, validation Accuracy : {:.2f}%'.format(i, trn_acc, val_acc))

    val_scores.append(val_acc)
    y_pred += (clf.predict_proba(x_test)[:, 1] / skf.n_splits)

# Mean Validation Score
print('Cross Validation Score : {:.2f}%'.format(np.mean(val_scores)))
```

```
0 Fold, train Accuracy : 83.37%, validation Accuracy : 83.79%
1 Fold, train Accuracy : 84.00%, validation Accuracy : 83.24%
2 Fold, train Accuracy : 83.95%, validation Accuracy : 83.70%
3 Fold, train Accuracy : 83.87%, validation Accuracy : 83.79%
4 Fold, train Accuracy : 83.96%, validation Accuracy : 83.58%
Cross Validation Score : 83.62%
```

```
# 확률을 라벨로 변경해줍니다.
y_pred = [0 if y < 0.5 else 1 for y in y_pred]
print('Average Blending Acc : {:.2f}%'.format(accuracy_score(y_test, y_pred)*100))
```

```
Average Blending Acc : 83.63%
```

- 단일 랜덤 포레스트보다 성능이 향상
- 각 폴드별로 학습된 서로 다른 모델을 앙상블 하는 효과

• Reference

- Validation 데이터가 필요한 이유 : https://3months.tistory.com/118
- Sklearn, KFold : https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
- Sklearn, StratifedKFold : https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#sklearn.model_selection.StratifiedKFold
- Sklearn, Compare with KFold, StratifedKFold : https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html#sphx-glr-auto-examples-model-selection-plot-cv-indices-py
- Sklearn, Cross Validation Score : https://www.google.com/url?q=http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html&sa=U&ved=0ahUKEwiGxeHhqubhAhUKV7wKHbFhDrcQFggEMAA&client=internal-uds-cse&cx=016639176250731907682:tjtqbvtvij0&usg=AOvVaw0rlHEJ1ltDaghFv1bvPeRO
- Sklearn, GridSearchCV : https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- Sklearn, Voting Classifier : https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
- Scikit-Optimize, Documentation : https://scikit-optimize.github.io