

파이썬 라이브러리를 활용한 머신러닝

사이킷런 최신 변경 사항

2019-04-23

박해선

O'REILLY®


사이킷런 핵심 개발자가 쓴 머신러닝과 데이터 과학 실무서



Introduction to Machine Learning with Python

파이썬 라이브러리를 활용한 머신러닝 [번역개정판]

사이킷런 최신 버전을 반영한 풀컬러 번역개정판

 한빛미디어
Hanbit Media, Inc.

안드레아스 뮐러, 세라 가이드 지음
박해선 옮김

예스24, 교보문고
절찬리 판매중!

완전 컬러!

OneHotEncoder

- 숫자 뿐만 아니라 문자열 열 인식
- 0.22 버전부터는 고유한 정수 값을 카테고리 사용합니다
 - `pd.get_dummies()` 사용할 필요 없음

표 4-4 문자열 범주형 특성과 숫자 특성을 가진 DataFrame

	범주형 특성	숫자 특성
0	양말	0
1	여우	1
2	양말	2
3	상자	1

In [12]:

```
from sklearn.preprocessing import OneHotEncoder
# sparse=False로 설정하면 OneHotEncoder가 희소 행렬이 아니라 넘파이 배열을 반환합니다
ohe = OneHotEncoder(sparse=False)
print(ohe.fit_transform(demo_df))
```

Out [12]:

```
[[1.  0.  0.  0.  0.  1.]
 [0.  1.  0.  0.  1.  0.]
 [0.  0.  1.  0.  0.  1.]
 [0.  1.  0.  1.  0.  0.]]
```

ColumnTransformer

- 판다스 데이터프레임이나 넘파이 배열의 열마다 다른 변환 적용
- 넘파이 배열은 열 인덱스로 지정
- `ct.named_transformers_.onehot` 처럼 참조
- 변환된 데이터셋의 열이 원본 데이터에서 어떤 열인지 모름(향후 개선 예정)

In [15]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler

ct = ColumnTransformer(
    [("scaling", StandardScaler(), ['age', 'hours-per-week']),
     ("onehot", OneHotEncoder(sparse=False),
      ['workclass', 'education', 'gender', 'occupation'])])
```

make_column_transformer

- 파이프라인을 만들어 주는 make_pipeline과 유사
- (열리스트, 변환기 체) 튜플 전달. 0.22 에서 (변환기객체, 열리스트)로 바뀔 예정

In [19]:

```
from sklearn.compose import make_column_transformer
ct = make_column_transformer(
    (['age', 'hours-per-week'], StandardScaler()),
    (['workclass', 'education', 'gender', 'occupation'], OneHotEncoder(sparse=False)))
```

KBinsDiscretizer

```
bins = np.linspace(-3, 3, 11)
which_bin = np.digitize(X, bins=bins)
from sklearn.preprocessing import OneHotEncoder
# 변환을 위해 OneHotEncoder를 사용합니다.
encoder = OneHotEncoder(sparse=False)
# encoder.fit은 which_bin에 나타난 유일한 값을 찾습니다.
encoder.fit(which_bin)
X_binned = encoder.transform(which_bin)
print(X_binned[:5])
```

Out[15]:

```
[[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
```

```
from sklearn.preprocessing import KBinsDiscretizer

kb = KBinsDiscretizer(n_bins=10, strategy='uniform')
kb.fit(X)

X_binned = kb.transform(X)

array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]])
```

cross_validate

- 다중 평가 수행 가능
 - `cross_validate(..., score=['accuracy', 'roc_auc'], ...)`
 - `cross_validate(..., score={'acc': 'accuracy', 'ra': 'roc_auc'}, ...)`
- `cross_val_score`는 `cross_validate`를 호출한 뒤 `test_score` 키만 전달

In [8]:

```
from sklearn.model_selection import cross_validate
res = cross_validate(logreg, iris.data, iris.target, cv=5,
                    return_train_score=True)
display(res)
```

Out [8]:

```
{'fit_time': array([0.001, 0.001, 0.001, 0.001, 0.001]),
 'score_time': array([0., 0., 0., 0., 0.]),
 'test_score': array([1.   , 0.967, 0.933, 0.9   , 1.   ]),
 'train_score': array([0.95  , 0.967, 0.967, 0.975, 0.958])}
```

Pipeline caching

- Pipeline, make_pipeline에 memory 매개변수 추가
- 그리드서치를 수행할 때 반복적인 전처리 단계를 피할 수 있음
- 디스크 직렬화에 대한 비용이 상대적으로 크기 때문에 간단한 그리드탐색에는 효과가 적음

```
pipe = Pipeline([('preprocessing', StandardScaler()), ('classifier', SVC())],  
                memory="cache_folder")
```


RepeatedKFold, RepeatedStratifiedKFold

- 분할 폴드 수 n_splits: 5
- 반복 횟수: n_repeats: 10

In [3]:

```
from sklearn.model_selection import RepeatedStratifiedKFold

rskfold = RepeatedStratifiedKFold(random_state=42)
scores = cross_val_score(logreg, iris.data, iris.target, cv=rskfold)

print("교차 검증 점수:\n", scores)
print("교차 검증 평균 점수: {:.3f}".format(scores.mean()))
```

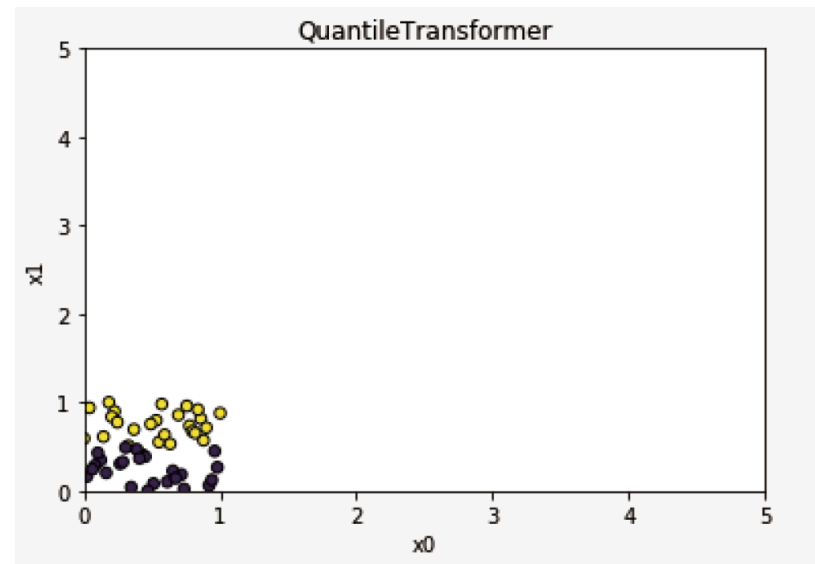
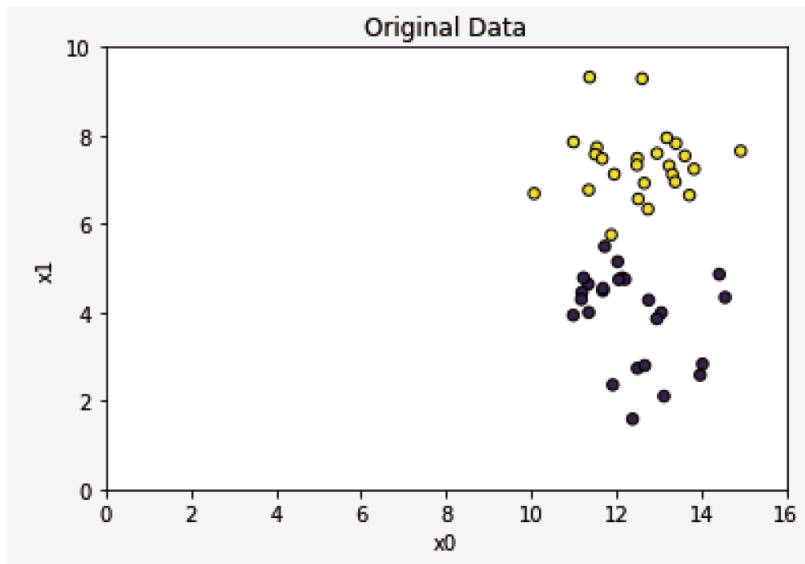
Out [3]:

```
교차 검증 점수:
[0.96666667 0.96666667 0.96666667 0.93333333 0.96666667 0.86666667
 0.96666667 0.96666667 0.93333333 0.96666667 1.          1.
 0.93333333 0.93333333 0.93333333 1.          0.96666667 0.96666667
 0.9          0.96666667 0.96666667 0.96666667 1.          0.9
 0.96666667 0.93333333 1.          0.96666667 0.96666667 0.93333333
 0.96666667 0.93333333 0.96666667 0.96666667 0.96666667 0.96666667
 0.93333333 0.93333333 0.96666667 1.          0.96666667 0.96666667
 0.86666667 1.          0.93333333 1.          0.96666667 1.
 0.93333333 0.9          ]
교차 검증 평균 점수: 0.957
```

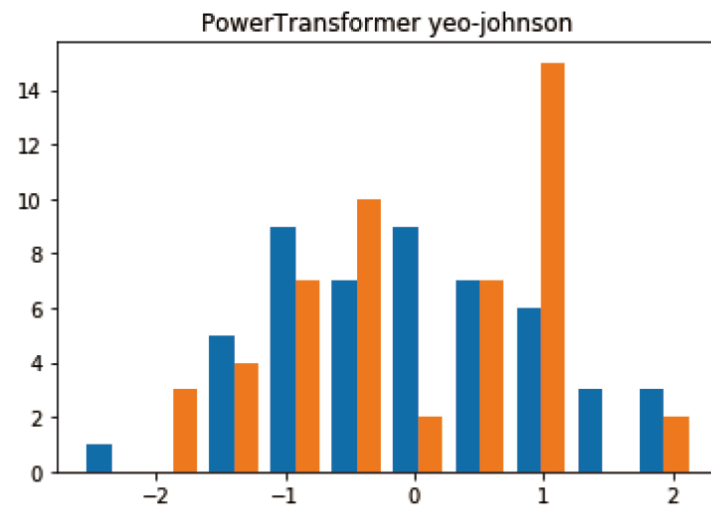
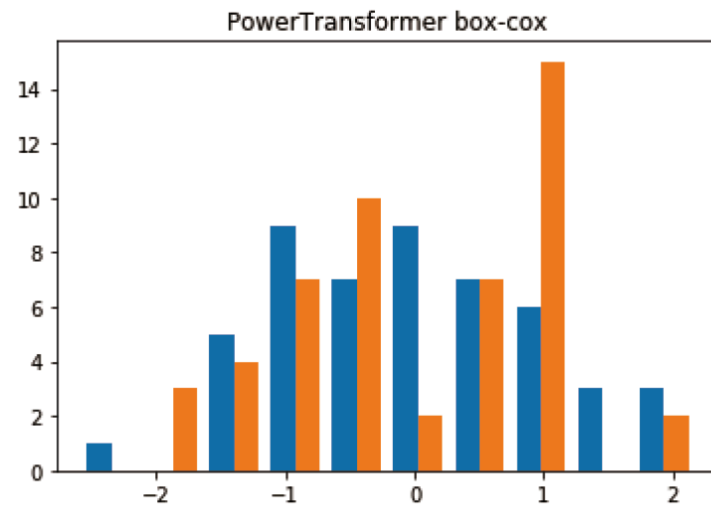
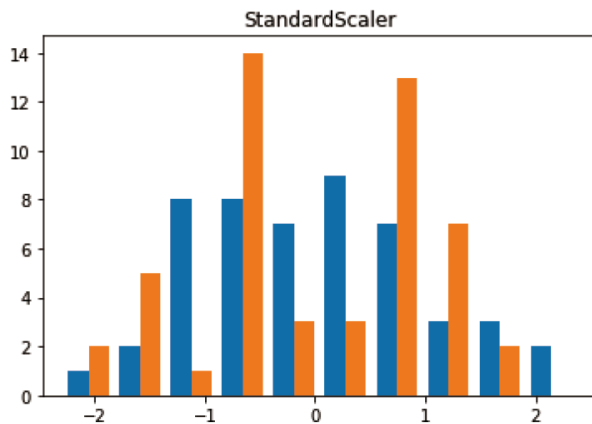
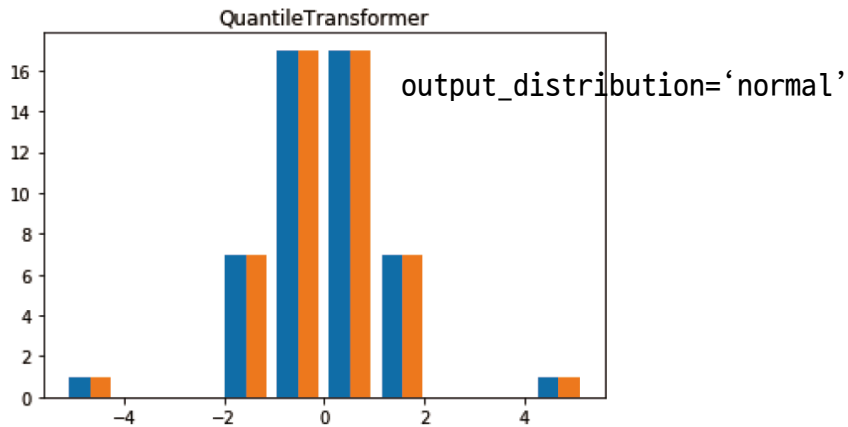
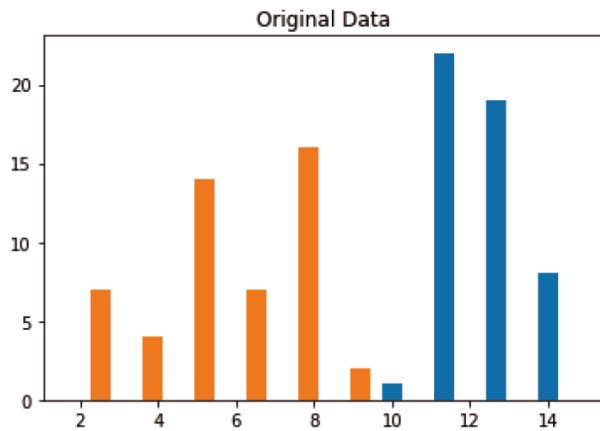
QuantileTransformer

- 1000개의 분위 사용(n_quantiles)
 - 0~1 사이로 압축
 - 이상치에 민감하지 않음

```
from sklearn.preprocessing import QuantileTransformer  
scaler = QuantileTransformer()  
X_trans = scaler.fit_transform(X)
```



PowerTransformer



BaggingClassifier

- Bootstrap aggregating, 중복을 허용한 랜덤 샘플링
 - oob_score 지원
- DecisionTreeClassifier(splitter='best')를 사용하면 랜덤 포레스트와 비슷
 - 0.20부터 파이프라인도 사용 가능

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier
bagging = BaggingClassifier(LogisticRegression(), n_estimators=100,
                             oob_score=True, n_jobs=-1, random_state=42)
bagging.fit(Xc_train, yc_train)
```

In [4]:

```
print("훈련 세트 정확도: {:.3f}".format(bagging.score(Xc_train, yc_train)))
print("테스트 세트 정확도: {:.3f}".format(bagging.score(Xc_test, yc_test)))
print("OOB 샘플의 정확도: {:.3f}".format(bagging.oob_score_))
```

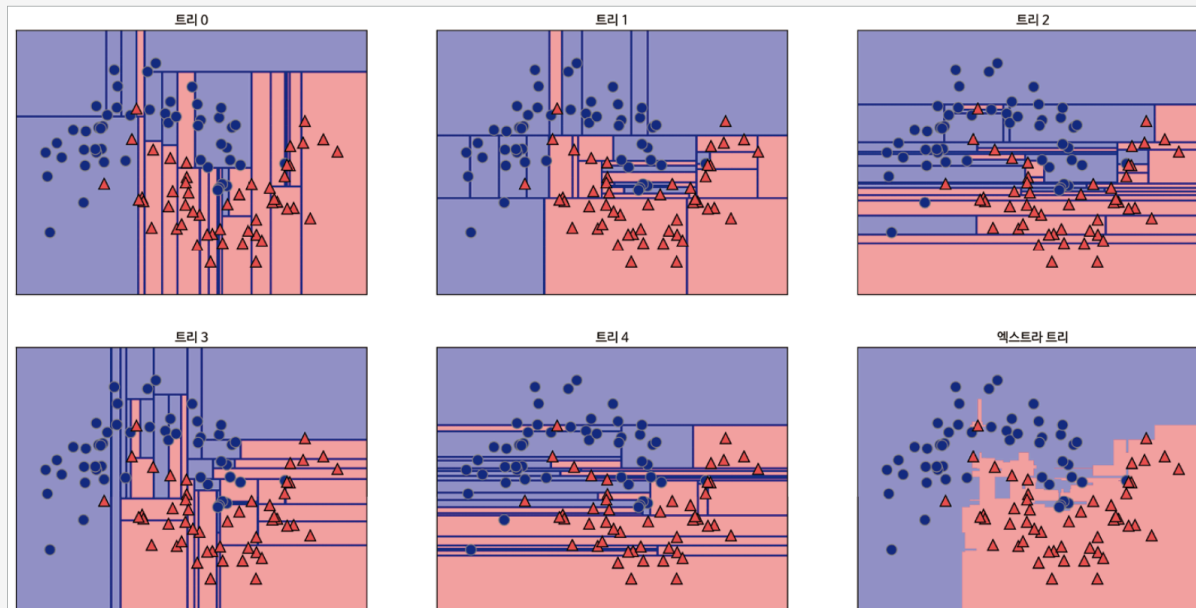
Out [4]:

```
훈련 세트 정확도: 0.962
테스트 세트 정확도: 0.958
OOB 샘플의 정확도: 0.948
```

ExtraTreesClassifier

- DecisionTreeClassifier(splitter='random') 사용
- 무작위로 선택한 후보 특성을 무작위로 분할
 - 부스트랩 샘플링 사용하지 않음
 - 편향을 늘리고 분산을 감소시킴

```
from sklearn.ensemble import ExtraTreesClassifier  
xtree = ExtraTreesClassifier(n_estimators=5, n_jobs=-1, random_state=0)  
xtree.fit(Xm_train, ym_train)
```



AdaBoostClassifier

- Adaptive Boosting
- 잘못 분류된 샘플의 가중치를 높여서 약한 학습기를 반복적으로 훈련
- DecisionTreeClassifier(max_depth=1), DecisionTreeRegressor(max_depth=3)
 - 순차적으로 학습해야 하므로 n_jobs 매개변수 지원하지 않음

```
from sklearn.ensemble import AdaBoostClassifier  
ada = AdaBoostClassifier(n_estimators=5, random_state=42)  
ada.fit(Xm_train, ym_train)
```



LogisticRegression

- 사이킷런 0.22버전에서 LogisticRegression의 solver 매개변수 기본값이 'liblinear'에서 'lbfgs'로 변경된다는 경고 출력
- 0.20 버전에서 LogisticRegression의 multi_class 매개변수에 'auto' 추가. 'auto'일 때 이진 분류이거나 solver가 'liblinear'일 경우에는 'ovr'을 선택하고 그 외에는 'multinomial' 선택
- multi_class 매개변수를 지정하지 않으면 0.22 버전부터 기본값이 'ovr'에서 'auto'로 변경된다는 경고 출력
- 대규모 데이터셋에 적합한 'saga' solver 추가

liblinear max_iter

- liblinear를 사용하는 LogisticRegression과 LinearSVC는 0.20 버전부터 알고리즘이 max_iter 반복 안에 수렴하지 않을 경우 반복 횟수를 증가하라는 경고 출력
- LogisticRegression의 max_iter 기본값은 100이고 LinearSVC의 max_iter 기본값은 1,000입니다.

RandomForestClassifier

- `n_estimators` 매개변수를 지정하지 않으면 0.22 버전부터 기본값이 10에서 100으로 바뀐다는 경고 출력

GradientBoosting

- 0.20 버전에서는 GradientBoostingClassifier와 GradientBoostingRegressor에 조기 종료를 위한 매개변수 `n_iter_no_change`와 `validation_fraction`이 추가
- 훈련 데이터에서 `validation_fraction`(기본값 0.1) 비율만큼 검증 데이터로 사용하여 `n_iter_no_change` 반복 동안 검증 점수가 향상되지 않으면 훈련 종료
- `n_iter_no_change`이 기본값 `None`이면 조기 종료 사용하지 않음

SVC

- 0.20 버전에서 gamma 매개변수 옵션에 'scale' 추가
- 'scale'은 $1/(X_train.shape[1] * X_train.std())$ 로 스케일 조정이 되지 않은 특성에서 더 좋은 결과 만듦
- 0.22 버전부터는 gamma 매개변수의 기본값이 'auto'에서 'scale'로 변경
- SVC 사용하기 전에 특성을 표준화 전처리하면 'scale'과 'auto'는 차이가 없습니다.

MLPClassifier

- solver가 'sgd' 또는 'adam'일 때 early_stopping을 'True'로 설정하면 연속된 검증 점수가 'tol' 값만큼 향상되지 않을 경우 반복을 자동으로 종료
- 0.20 버전에서는 조기 종료하기 위해 검증 점수가 향상되지 않는 반복 횟수의 기준을 지정하는 n_iter_no_change 매개변수 추가
- 검증 데이터의 비율은 validation_fraction 매개변수에서 지정하며 기본값은 0.1(훈련 데이터의 10%)

AgglomerativeClustering

- ward : 기본값인 ward 연결linkage은 모든 클러스터 내의 분산을 가장 작게 증가시키는 두 클러스터를 합칩. 크기가 비교적 비슷한 클러스터
- average : average 연결은 클러스터 포인트 사이의 평균 거리가 가장 짧은 두 클러스터를 합칩
- complete : complete 연결(최대 연결이라고도 합니다)은 클러스터 포인트 사이의 최대 거리가 가장 짧은 두 클러스터를 합칩
- single : (0.20 추가) 클러스터 포인트 사이의 최소 거리가 가장 짧은 두 클러스터를 합칩

DBSCAN

- 0.20 버전에서 algorithm에 상관없이 n_jobs 매개변수 사용 가능

Default Fold

- 0.22 버전부터 `cross_val_score`의 교차 검증의 기본값은 3-겹 교차 검증에서 5-겹 교차 검증으로 바뀜
- `GridSearchCV` 클래스도 0.22 버전부터 3-겹 교차 검증에서 5-겹 교차 검증으로 바뀜

GridSearchCV

- 0.22 버전부터는 GridSearchCV의 iid 매개변수 기본값이 True에서 False로 변경. 0.24 버전에서는 삭제될 예정임.
- iid 매개변수가 True이면 독립 동일 분포라고 가정하고 테스트 세트의 샘플 수로 폴드의 점수를 가중 평균
- False로 지정하면 단순한 폴드 점수의 평균이므로 기본 교차 검증과 동작 방식이 같음.
- 0.20 버전에서 iid 매개변수가 기본값일 때 가중 평균과 단순 평균의 차이가 10^{-4} 이상이면 경고.
- 성능 향상을 위해 return_train_score 매개변수의 기본값이 0.21 버전부터 False로 변경. cv_results_ 속성에 훈련 폴드의 점수를 포함시키려면 True로 설정.

GridSearchCV

- 그리드서치와 모델에 모두 `n_jobs` 지정 가능
- `joblib`에서 새로운 멀티프로세싱 라이브러리 `loky` 사용
- 충분한 테스트가 필요함

SimpleImputer

- sklearn.preprocessing.Imputer → sklearn.impute.SimpleImputer
- missing_values의 기본값: 'NaN' → np.nan
- strategy 옵션: mean, median, most_frequent, constant(추가)
- fill_value(추가)

```
>>> import numpy as np
>>> from sklearn.impute import SimpleImputer
>>> simr = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> simr = simr.fit(df.values)
>>> imputed_data = simr.transform(df.values)
>>> imputed_data
array([[ 1. ,  2. ,  3. ,  4. ],
       [ 5. ,  6. ,  7.5,  8. ],
       [10. , 11. , 12. ,  6. ]])
```

OrdinalEncoder

- 범주형 데이터를 정수 레이블로 인코딩
- LabelEncoder는 타깃 데이터를 정수 레이블로 인코딩

classification_report

- "macro" 평균은 클래스별 f1-점수에 가중치를 주지 않고 클래스 크기에 상관없이 모든 클래스를 같은 비중으로 다룸
- "weighted" 평균은 클래스별 샘플 수로 가중치를 두어 f1-점수의 평균을 계산
- "micro" 평균은 모든 클래스의 거짓 양성(FP), 거짓 음성(FN), 진짜 양성(TP)의 총 수를 헤아린 다음 정밀도, 재현율, f1-점수를 계산

```
print(classification_report(y_test, pred_dummy,  
                             target_names=["9 아님", "9"]))
```

	precision	recall	f1-score	support
9 아님	0.90	0.90	0.90	403
9	0.11	0.11	0.11	47
micro avg	0.82	0.82	0.82	450
macro avg	0.50	0.50	0.50	450
weighted avg	0.81	0.82	0.82	450

감사합니다.