




ML-C6

22.07.14



알고리즘체인과 파이프라인

대부분의 머신러닝 애플리케이션은 여러단계의 처리과정
과 머신러닝 모델이 연결되어있다.

데이터 변환 과정과 머신러닝 모델을 쉽게 연결해주는
Pipeline 파이썬클래스를 이용해보기.

Review-> SVM(cancer_dataset , MinMaxScaler)



```
cancer = load_breast_cancer()
```

```
X_train, X_test, y_train, y_test = train_test_split(cancer.data,  
cancer.target,random_state=0)
```

```
scaler = MinMaxScaler().fit(X_train)  
X_trained_scale = scaler.transform(X_train)  
X_test_scale = scaler.transform(X_test)
```

```
svm = SVC().fit(X_trained_scale, y_train)  
svm.score(X_test_scale,y_test)
```

```
0.951048951048951
```

Data_Preprocessing / Parameter_Selection



```
cancer = load_breast_cancer()
```

```
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=0)
```

```
scaler = MinMaxScaler().fit(X_train)
```

```
X_train_scaled = scaler.transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

0.951048951048951

```
svm = SVC().fit(X_train_scaled, y_train)
```

```
svm.score(X_test_scaled, y_test)
```

```
cancer = load_breast_cancer()
```

```
X_train, X_test, y_train, y_test = train_test_split(cancer.data,  
cancer.target, random_state=0)
```

```
scaler = MinMaxScaler().fit(X_train)
```

```
X_train_scaled = scaler.transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

```
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=5)
```

```
grid.fit(X_train_scaled, y_train)
```

```
print(grid.best_score_)
```

```
print(grid.score(X_test_scaled, y_test))
```

```
print(grid.best_params_)
```

Train 75에 해당하는 feature값에 0과 1 사이로 Scaling시키고, 그리드 서치(테스트셋에서 75대 25를 통해, 검증스코어가 높은 파라미터값(c, gamma)를 찾고 이를 best_parameter로 정하고 test셋을 이에 적용 시킨다.

Train 75에서 이미 minmax로 스케일링 처리되었으므로, 스케일조정된 데이터를 검증세트로 활용했으므로, 이건 학습된 훈련세트를 검증세트에 활용했다는 것이다. 보통은 train따로 검증세트는 추후에 들어온 새로운데이터 이기 때문이다.

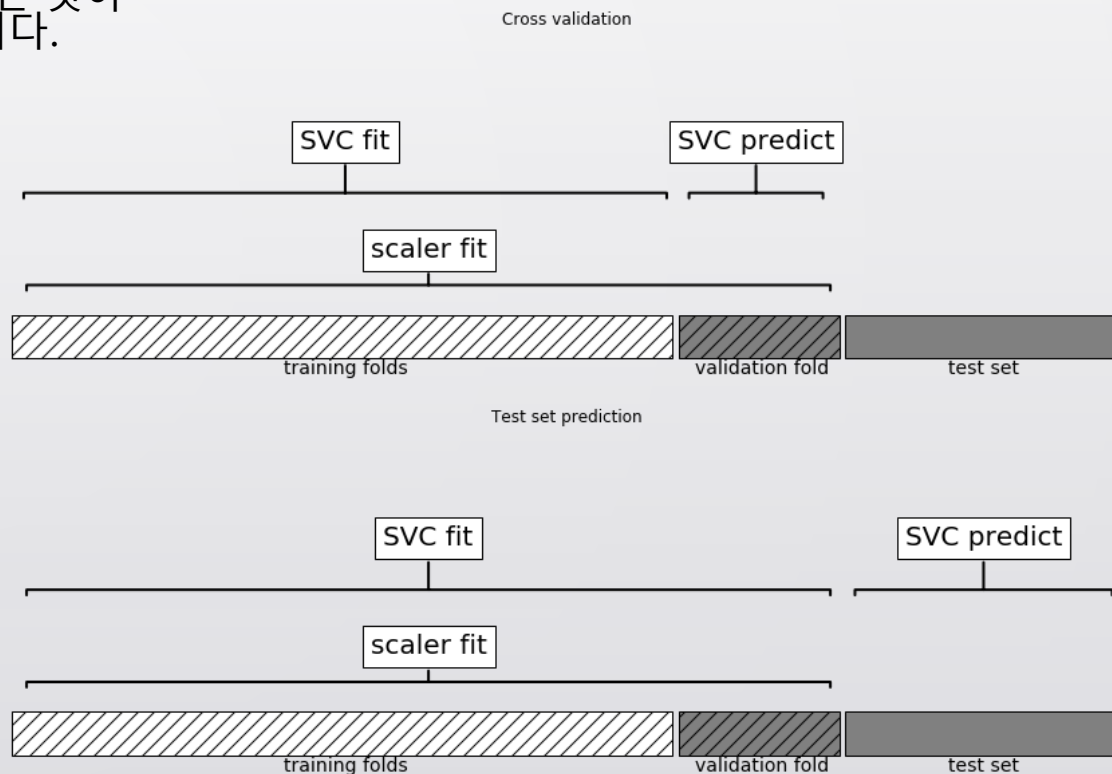
0.9812206572769953 0.972027972027972 {'C': 1, 'gamma': 1}

Data_Preprocessing / Parameter_Selection



Train 75에 해당하는 featur값에 0과1사이로 Scaling시키고, 그리드 서치(테스트셋에서 75대 25를 통해, 검증스코어가 높은 파라미터값(c,gamma)를 찾고 이를 best_parameter로 정하고 test셋을 이에 적용 시킨다.

Train 75에서 이미 minmax로 스케일링 처리되었으므로, 스케일조정된 데이터를 검증세트로 활용했으므로, 이걸 학습된 훈련세트를 검증세트에 활용했다는 것이다. 보통은 train따로 검증세트는 추후에 들어온 새로운데이터 이기때문이다.



Pipeline 구축하기



```
from sklearn.pipeline import Pipeline
cancer = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(cancer.data,
cancer.target, random_state=0)

pipe = Pipeline([("scaler", MinMaxScaler()), ("svm", SVC())])

pipe.fit(X_train, y_train)

pipe.score(X_test, y_test)
```

```
0.951048951048951
```


GridSearch에 Pipeline 적용하기

```
from sklearn.pipeline import Pipeline
```

```
param_grid = {'svm__C': [0.001,0.01,0.1,1,10,100], 'svm__gamma': [0.001,0.01,0.1,1,10,100]}
```

```
cancer = load_breast_cancer()
```

```
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=0)
```

```
pipe = Pipeline([("scaler", MinMaxScaler()), ("svm", SVC())])
```

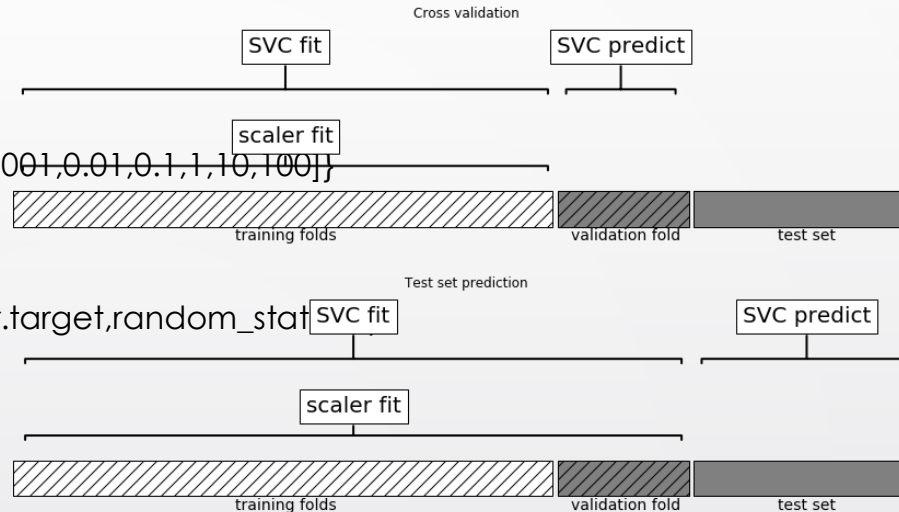
```
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5)
```

```
grid.fit(X_train, y_train)
```

```
print(grid.best_score_)
```

```
print(grid.score(X_test, y_test))
```

```
print(grid.best_params_)
```



이 경우, 교차검증 시마다 스케일링이 훈련폴드에 매번 적용되어, 매개변수 검색과정에서 폴드의 정보가 누설되지 않는다.

```
0.9812206572769953 0.972027972027972 {'svm__C': 1, 'svm__gamma': 1}
```

정보누설에 대한 예시(정규분포로부터 독립적 추출한 10000개의 특성을 가진 샘플100개)



```
from sklearn.feature_selection import SelectPercentile, f_regression
```

```
rnd = np.random.RandomState(seed=0)
```

```
X = rnd.normal(size=(100,10000))
```

```
y = rnd.normal(size=(100,))
```

```
select = SelectPercentile(score_func=f_regression, percentile = 5).fit(X,y)
```

```
X_selected = select.transform(X)
```

```
print(X_selected.shape)
```

```
np.mean(cross_val_score(Ridge(),X_selected,y,cv=5))
```

10000개의 특성을 500개로 줄인 후에, 교차검증을

실시한다는 것은 타겟을 나누기도전에 전체 데이터의
연관성을 부여할 수 있다는 것이 함정이다.

정보누설에 대한 예시(정규분포로부터 독립적 추출한 10000개의 특성을 가진 샘플100개)



```
from sklearn.feature_selection import SelectPercentile, f_regression
```

```
rnd = np.random.RandomState(seed=0)
```

```
X = rnd.normal(size=(100,10000))
```

```
y = rnd.normal(size=(100,))
```

```
pipe = Pipeline([("select",SelectPercentile(score_func=f_regression, percentile =  
5)),("ridge",Ridge())])
```

```
np.mean(cross_val_score(pipe,X,y,cv=5))
```

-0.2465542238495281

10000개의 특성을 500개로 줄인 후에, 교차검증을

실시한다는 것은 타겟을 나누기도전에 전체 데이터의
연관성을 부여할 수 있다는 것이 함정이다.

Pipeline의 조건!



10000개의 특성을 500개로 줄인 후에, 교차검증을

실시한다는 것은 타겟을 나누기도전에 전체 데이터의
연관성을 부여할 수 있다는 것이 함정이다.

파이프라인.fit 메소드가 실행되는 동안, 파이프라인은
각단계에서 이전 단계의 transform의 출력을 입력으로 받아
fit과 transform메소드를 차례로 호출한다.

마지막단계는 fit만 호출한다.

-> pipeline.steps는 튜플의 리스트라서
pipeline.steps[0][1]은 첫번째 추정기이고
pipeline.steps[1][1]은 두번째 추정기가 되는 식.

```
def fit(self, X,y):
    X_transformed = X
    for name, estimator in self.steps[:-1]:
        #마지막 단계를 제외하고 fit과 transform을 반복한다.
        X_transformed = estimator.fit_transform(X_transformed, y)
    self.steps[-1][1].fit(X_transformed,y)
    return self
```

```
def predict(self, X):
    X_transformed = X
    for step in self.steps[:-1]:
        #마지막 단계를 제외하고 fit과 transform을 반복한다.
        X_transformed = step[1].transform(X_transformed, y)
    return self.steps[-1][1].predict(X_transformed)
```

Pipeline, Make_Pipeline



```
pipe_short = make_pipeline(StandardScaler(),PCA(n_components=2),StandardScaler())  
pipe_short.fit(cancer.data)
```

```
print(pipe_short.steps)  
comp = pipe_short.named_steps["pca"].components_  
print(comp.shape)
```

```
[('standardscaler-1', StandardScaler(copy=True, with_mean=True, with_std=True)),  
(('pca', PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,  
svd_solver='auto', tol=0.0, whiten=False))), ('standardscaler-2',  
StandardScaler(copy=True, with_mean=True, with_std=True))]
```

```
(2, 30)
```

GridSearch안의 파이프라인 속성에 접근하기.



```
pipe_short = make_pipeline(StandardScaler(),LogisticRegression())

param_grid = {'logisticregression__C' : [0.01,0.1,1,10,100]}

X_train, X_test, y_train, y_test = train_test_split(cancer.data,cancer.target,random_state=4)

grid = GridSearchCV(pipe_short,param_grid,cv=5)
grid.fit(X_train,y_train)

print(grid.best_estimator_.named_steps["logisticregression"].coef_)
```

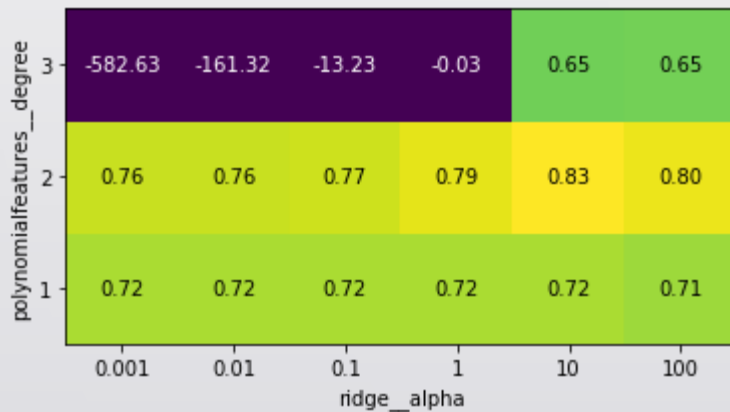
```
[[-0.38856355 -0.37529972 -0.37624793 -0.39649439 -
 0.11519359  0.01709608 -0.3550729 -0.38995414 -
 0.05780518  0.20879795 -0.49487753 -0.0036321 -
 0.37122718 -0.38337777 -0.04488715  0.19752816
 0.00424822 -0.04857196  0.21023226  0.22444999 -
 0.54669761 -0.52542026 -0.49881157 -0.51451071 -
 0.39256847 -0.12293451 -0.38827425 -0.4169485 -
 0.32533663 -0.13926972]]
```

전처리와 모델의 매개변수를 위한 그리드 서치

```
boston = load_boston()
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target, random_state=0)
pipe_short = make_pipeline(StandardScaler(), PolynomialFeatures(), Ridge())
param_grid = {'polynomialfeatures__degree': [1, 2, 3], 'ridge__alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
```

```
grid = GridSearchCV(pipe_short, param_grid, cv=5, n_jobs=-1)
grid.fit(X_train, y_train)
```

```
mglearn.tools.heatmap(grid.cv_results_['mean_test_score'].reshape(3, -1),
                        xlabel="ridge__alpha", xticklabels=param_grid['ridge__alpha'],
                        ylabel="polynomialfeatures__degree", yticklabels=param_grid['polynomialfeatures__degree'],
                        vmin=0)
```



```
{'polynomialfeatures__degree': 2, 'ridge__alpha': 10}
```

```
0.7683045464100136
```


모델 선택을 위한 그리드 서치.

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

```
param_grid = [  
    {'classifier': [SVC()], 'preprocessing': [StandardScaler()], 'classifier__gamma': [0.001, 0.01, 0.1, 1, 10, 100],  
     'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100]}, {'classifier':  
    [RandomForestClassifier(n_estimators=100)], 'preprocessing': [None], 'classifier__max_features'  
     : [1, 2, 3]}]
```

```
cancer = load_breast_cancer()  
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=0)  
grid = GridSearchCV(pipe, param_grid, cv=5)  
grid.fit(X_train, y_train)
```

```
print(grid.best_params_)  
print(grid.best_score_)  
print(grid.score(X_test, y_test))
```

```
{'classifier': SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,  
decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf', max_iter=-1,  
probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False),  
'classifier__C': 10, 'classifier__gamma': 0.01, 'preprocessing':  
StandardScaler(copy=True, with_mean=True, with_std=True)} 0.9859154929577465  
0.9790209790209791
```


중복계산피하기.



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

중복계산이 불가피할 경우, (전처리 과정 : PCA , NMF와
같이 시간이 오래 걸리는 것) 경우 Pipeline에서
memory옵션을 지정해주면 시간 절약을 할 수 있다.