wines_knn_cross_val

November 26, 2021

1 Validation croisée (cross-validation)

Il est acquis qu'un modèle doit être évalué sur une base de test différente de celle utilisée pour l'apprentissage. Mais la performance est peut-être juste l'effet d'une aubaine et d'un découpage particulièrement avantageux. Pour être sûr que le modèle est robuste, on recommence plusieurs fois. On appelle cela la validation croisée ou cross validation.

[1]:
```
from pyquickhelper.helpgen import NbImage
NbImage('images/cross.png', width=300)
```

[1]:

On découpe la base de données en cinq segments de façon aléatoire. On en utilise 4 pour l'apprentissage et 1 pour tester. On recommander 5 fois. Si le modèle est robuste, les cinq de scores de test seront sensiblement égaux.

[2]:
```
%matplotlib inline
```

[3]:
```
from papierstat.datasets import load_wines_dataset
df = load_wines_dataset()
X = df.drop(['quality', 'color'], axis=1)
y = df['quality']
df.head()
```

[3]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	\
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	\
0	11.0	34.0	0.9978	3.51	0.56	
1	25.0	67.0	0.9968	3.20	0.68	

```
2                15.0                 54.0   0.9970  3.26       0.65
3                17.0                 60.0   0.9980  3.16       0.58
4                11.0                 34.0   0.9978  3.51       0.56

   alcohol  quality color
0      9.4        5   red
1      9.8        5   red
2      9.8        5   red
3      9.8        6   red
4      9.4        5   red
```

On utilise un modèle des plus proches voisins.

[4]:
```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=1)
```

Nous allons utiliser la fonction cross_val_score.

[5]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(knn, X, y, cv=5)
```

[5]:
```
array([-0.83897083, -0.4670711 , -0.59014921, -0.38119203, -0.77196458])
```

Le score par défaut est R^2 :

[6]:
```
from sklearn.metrics import make_scorer, r2_score
cross_val_score(knn, X, y, cv=5, scoring=make_scorer(r2_score))
```

[6]:
```
array([-0.83897083, -0.4670711 , -0.59014921, -0.38119203, -0.77196458])
```

Si on souhaite utiliser score un autre score :

[7]:
```
from sklearn.metrics import mean_squared_error
cross_val_score(knn, X, y, cv=5, scoring=make_scorer(mean_squared_error))
```

[7]:
```
array([1.21615385, 1.21230769, 1.27328714, 1.14857583, 1.13702848])
```

Ou plusieurs à la fois :

[8]:
```
from sklearn.model_selection import cross_validate
cross_validate(knn, X, y, cv=5, scoring=dict(r2=make_scorer(r2_score),␣
↪e2=make_scorer(mean_squared_error)),
               return_train_score=False)
```

[8]:
```
{'fit_time': array([0.00997543, 0.01396203, 0.00897694, 0.00796676,
0.00897527]),
 'score_time': array([0.02692699, 0.03490806, 0.03518105, 0.0289228 ,
0.03111148]),
 'test_r2': array([-0.83897083, -0.4670711 , -0.59014921, -0.38119203,
-0.77196458]),
 'test_e2': array([1.21615385, 1.21230769, 1.27328714, 1.14857583, 1.13702848])}
```

On obtient bien les mêmes résultats mais ils sont bien différents de ceux obtenus avec train_est_split et reproduits ci-dessous.

[9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
knn.fit(X_train, y_train)
```

```
prediction = knn.predict(X_test)
r2_score(y_test, prediction)
```

[9]:
```
-0.07380837441932231
```

Ca doit mettre la **puce à l'oreille**. De plus, étonnamment, le score R^2 est identique pour les tirages si on réexecute le code une seconde fois pour la validation croisée alors qu'il est différent pour une seconde répartition apprentissage test :

[10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y)
knn.fit(X_train, y_train)
prediction = knn.predict(X_test)
r2_score(y_test, prediction)
```

[10]:
```
-0.06912470376051227
```

Les résultats sont rigoureusement identique pour la validation croisée.

[11]:
```
cross_validate(knn, X, y, cv=5, scoring=dict(r2=make_scorer(r2_score),␣
↪e2=make_scorer(mean_squared_error)),
               return_train_score=False)
```

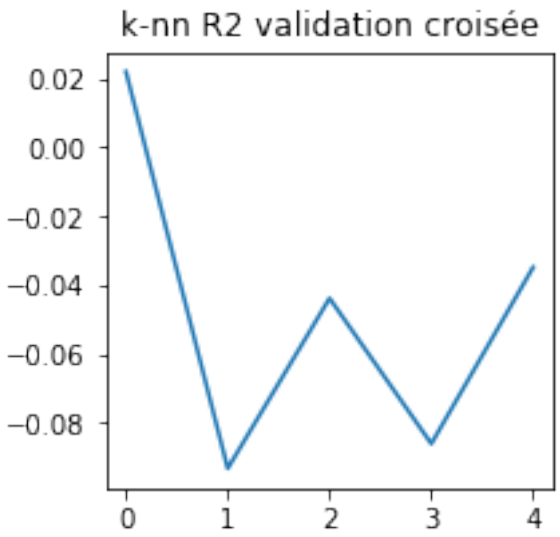
[11]:
```
{'fit_time': array([0.01196742, 0.00704718, 0.00997663, 0.00997043,
0.00697613]),
 'score_time': array([0.02892447, 0.02785683, 0.03490329, 0.03490925,
0.03045082]),
 'test_r2': array([-0.83897083, -0.4670711 , -0.59014921, -0.38119203,
-0.77196458]),
 'test_e2': array([1.21615385, 1.21230769, 1.27328714, 1.14857583, 1.13702848])}
```

C'est quelque peu **suspect**, très suspect en fait, en statistique, c'est quasi miraculeux pour un nombre aussi volatile. Cela ne peut être dû au fait que la fonction fait exactement les mêmes découpages. Mettons un peu plus d'aléatoire :

[12]:
```
from sklearn.model_selection import StratifiedKFold
from time import perf_counter
res = cross_validate(knn, X, y,
                     scoring=dict(r2=make_scorer(r2_score),␣
↪e2=make_scorer(mean_squared_error)),
                     return_train_score=False,
                     cv=StratifiedKFold(n_splits=5, random_state=int(perf_counter ()*100),␣
↪shuffle=True))
res
```

[12]:
```
{'fit_time': array([0.01196885, 0.00897121, 0.00782824, 0.01192856,
0.00897884]),
 'score_time': array([0.0309217 , 0.02644658, 0.02895927, 0.03191495,
0.02892423]),
 'test_r2': array([ 0.02195033, -0.09317781, -0.04380849, -0.08601952,
-0.03480481]),
 'test_e2': array([0.74807988, 0.83461538, 0.79692308, 0.82588598, 0.78643022])}
```

[13]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(3, 3))
ax.plot(res['test_r2'])
ax.set_title('k-nn R2 validation croisée');
```



On retrouve les mêmes scores que pour train_test_split. Comment l'interpréter ? La raison la plus probable est que la validation croisée implémenté par *scikit-learn* n'est par défaut pas aléatoire. Cela explique qu'on retrouve les mêmes résultats sur deux exécutions. Il reste à expliquer le fait que les chiffres sont nettement mauvais pour le premier code et meilleur pour ce second code.

Et si les vins n'étaient pas mélangés dans la base avec des vins rouges au début et blancs vers la fin ?

[14]:
```
dfi = df.reset_index(drop=False)
import pandas
pandas.concat([dfi[['index', 'color']].head(), dfi[['index', 'color']].tail()])
```

[14]:

	index	color
0	0	red
1	1	red
2	2	red
3	3	red
4	4	red
6492	6492	white
6493	6493	white
6494	6494	white
6495	6495	white
6496	6496	white

[15]:
```
dfi[['index', 'color']].groupby('color').min()
```

[15]:

color	index
red	0
white	1599

[16]:
```
dfi[['index', 'color']].groupby('color').max()
```

[16]:

color	index

```
red     1598
white   6496
```

Les éléments sont clairements triés par couleur et la validation croisée par défaut découpe selon cet ordre. Cela signifie presque que le modèle essaye de prédire la note d'un vin rouge en s'appuyant sur des vins blancs et cela ne marche visiblement pas. La validation croisée ne retourne pas de modèle mais cela peut être contourné avec GridSearchCV.

[17]:
```
from sklearn.model_selection import GridSearchCV
cvgrid = GridSearchCV(estimator=knn, param_grid={},
                      scoring=dict(r2=make_scorer(r2_score),␣
 ↪e2=make_scorer(mean_squared_error)),
                      return_train_score=False, refit='r2',
                      cv=StratifiedKFold(n_splits=5, random_state=int(perf_counter ()*100),␣
 ↪shuffle=True))
```

[18]:
```
cvgrid.fit(X, y)
```

[18]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=5289, shuffle=True),
             error_score='raise-deprecating',
             estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30,
                                           metric='minkowski',
                                           metric_params=None, n_jobs=None,
                                           n_neighbors=1, p=2,
                                           weights='uniform'),
             iid='warn', n_jobs=None, param_grid={}, pre_dispatch='2*n_jobs',
             refit='r2', return_train_score=False,
             scoring={'e2': make_scorer(mean_squared_error),
                      'r2': make_scorer(r2_score)},
             verbose=0)
```

[19]:
```
cvgrid.cv_results_
```

[19]:
```
{'mean_fit_time': array([0.00996728]),
 'std_fit_time': array([0.00201074]),
 'mean_score_time': array([0.02833037]),
 'std_score_time': array([0.0041993]),
 'params': [{}],
 'split0_test_r2': array([-0.10356939]),
 'split1_test_r2': array([-0.03272558]),
 'split2_test_r2': array([-0.0115673]),
 'split3_test_r2': array([-0.07183643]),
 'split4_test_r2': array([-0.05103704]),
 'mean_test_r2': array([-0.05415835]),
 'std_test_r2': array([0.03175736]),
 'rank_test_r2': array([1]),
 'split0_test_e2': array([0.84408602]),
 'split1_test_e2': array([0.78846154]),
 'split2_test_e2': array([0.77230769]),
 'split3_test_e2': array([0.81510015]),
 'split4_test_e2': array([0.79876638]),
 'mean_test_e2': array([0.80375558]),
 'std_test_e2': array([0.02452202]),
 'rank_test_e2': array([1])}
```

[20]:
```
cvgrid.best_estimator_
```

[20]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                    weights='uniform')
```

[21]:
```

```

[22]:
```

```