wines_knn_split_strat

November 26, 2021

1 Découpage stratifié apprentissage / test

Lorsqu'une classe est sous-représentée, il y a peu de chances que la répartition apprentissage test conserve la distribution des classes.

[1]:
```
%matplotlib inline
```

[2]:
```
from papierstat.datasets import load_wines_dataset
df = load_wines_dataset()
X = df.drop(['quality', 'color'], axis=1)
y = df['quality']
```

On divise en base d'apprentissage et de test avec la fonction train_test_split.

[3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

[4]:
```
import pandas
ys = pandas.DataFrame(dict(y=y_train))
ys['base'] = 'train'
ys2 = pandas.DataFrame(dict(y=y_test))
ys2['base'] = 'test'
ys = pandas.concat([ys, ys2])
ys['compte'] = 1
piv = ys.groupby(['base', 'y'], as_index=False).count().pivot('y', 'base', 'compte')
piv['ratio'] = piv['test'] / piv['train']
piv
```

[4]:

y	base test	train	ratio
3	13.0	17.0	0.764706
4	54.0	162.0	0.333333
5	539.0	1599.0	0.337086
6	713.0	2123.0	0.335846
7	267.0	812.0	0.328818
8	39.0	154.0	0.253247
9	NaN	5.0	NaN

On voit le ratio entre les deux classes est à peu près égal à 1/3 sauf pour les notes sous-représentées. On utilise une répartition stratifiée : la distribution d'une variable, les labels, sera la même dans les bases d'apprentissages et de de tests. On s'inspire de l'exemple StratifiedShuffleSplit.

[5]:
```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.33)
train_index, test_index = list(split.split(X, y))[0]
len(train_index), len(test_index)
```

[5]:
```
(4352, 2145)
```

[6]:
```
X_train, y_train = X.iloc[train_index, :], y[train_index]
X_test, y_test = X.iloc[test_index, :], y[test_index]
y_train.shape, y_test.shape
```

[6]:
```
((4352,), (2145,))
```

[7]:
```
ys = pandas.DataFrame(dict(y=y_train))
ys['base'] = 'train'
ys2 = pandas.DataFrame(dict(y=y_test))
ys2['base'] = 'test'
ys = pandas.concat([ys, ys2])
ys['compte'] = 1
piv = ys.groupby(['base', 'y'], as_index=False).count().pivot('y', 'base', 'compte')
piv['ratio'] = piv['test'] / piv['train']
piv
```

[7]:

base	test	train	ratio
y			
3	10	20	0.500000
4	71	145	0.489655
5	706	1432	0.493017
6	936	1900	0.492632
7	356	723	0.492393
8	64	129	0.496124
9	2	3	0.666667

Le ratio entre les classes est identique, la classe test contient deux fois moins d'invidivu et c'est vrai pour toutes les classes excepté pour la classe 9 qui contient si peu d'éléments que c'est impossible.

[8]:
```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=1)
knn.fit(X_train, y_train)
```

[8]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=1, p=2,
          weights='uniform')
```

[9]:
```
prediction = knn.predict(X_test)
```

[10]:
```
from sklearn.metrics import r2_score
r2_score(y_test, prediction)
```

[10]:
```
-0.1007330402006481
```

Cela n'améliore pas la qualité du modèle mais on est sûr que les classes sous-représentées sont mieux gérées par cette répartition aléatoire stratifiée.

[11]:
```
```

[12]: