wines_multi

November 26, 2021

1 Classification multi-classe

On cherche à prédire la note d'un vin avec un classifieur multi-classe.

[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']
```

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

[4]:
```
from sklearn.linear_model import LogisticRegression
clr = LogisticRegression()
clr.fit(X_train, y_train)
```

[4]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

[5]:
```
import numpy
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[5]:
```
53.84615384615385
```

On regarde la matrice de confusion.

[6]:
```
from sklearn.metrics import confusion_matrix
import pandas
pandas.DataFrame(confusion_matrix(y_test, clr.predict(X_test)))
```

[6]:

	0	1	2	3	4	5	6
0	0	0	6	0	0	0	0
1	0	0	39	14	1	0	0
2	0	0	338	208	2	0	0
3	0	0	195	517	17	0	0
4	0	0	19	200	20	0	0
5	0	0	2	38	8	0	0
6	0	0	0	1	0	0	0

On l'affiche différemment avec le nom des classes.

[7]:
```
conf = confusion_matrix(y_test, clr.predict(X_test))
dfconf = pandas.DataFrame(conf)
labels = list(clr.classes_)
if len(labels) < dfconf.shape[1]:
    labels += [9]  # La classe 9 est très représentée, elle est parfois absente en␣
 ↪train.
elif len(labels) > dfconf.shape[1]:
    labels = labels[:dfconf.shape[1]] # ou l'inverse
dfconf.columns = labels
dfconf.index = labels
dfconf
```

[7]:

	3	4	5	6	7	8	9
3	0	0	6	0	0	0	0
4	0	0	39	14	1	0	0
5	0	0	338	208	2	0	0
6	0	0	195	517	17	0	0
7	0	0	19	200	20	0	0
8	0	0	2	38	8	0	0
9	0	0	0	1	0	0	0

Pas extraordinaire. On applique la stratégie OneVsRestClassifier.

[8]:
```
from sklearn.multiclass import OneVsRestClassifier
clr = OneVsRestClassifier(LogisticRegression())
clr.fit(X_train, y_train)
```

[8]:
```
OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None,
dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
          n_jobs=1)
```

[9]:
```
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[9]:
```
53.784615384615385
```

Le modèle logistique régression multi-classe est équivalent à la stratégie *OneVsRest*. Voyons l'autre.

[10]:
```
from sklearn.multiclass import OneVsOneClassifier
clr = OneVsOneClassifier(LogisticRegression())
clr.fit(X_train, y_train)
```

[10]:
```
OneVsOneClassifier(estimator=LogisticRegression(C=1.0, class_weight=None,
dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
          n_jobs=1)
```

[11]:
```
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[11]:
```
53.47692307692308
```

[12]:
```
conf = confusion_matrix(y_test, clr.predict(X_test))
dfconf = pandas.DataFrame(conf)
labels = list(clr.classes_)
if len(labels) < dfconf.shape[1]:
    labels += [9]  # La classe 9 est très représentée, elle est parfois absente en␣
 ↪train.
elif len(labels) > dfconf.shape[1]:
    labels = labels[:dfconf.shape[1]] # ou l'inverse
dfconf.columns = labels
dfconf.index = labels
dfconf
```

[12]:

	3	4	5	6	7	8	9
3	0	0	6	0	0	0	0
4	0	0	38	15	1	0	0
5	0	0	335	208	5	0	0
6	0	0	197	491	41	0	0
7	0	0	20	176	43	0	0
8	0	0	1	34	13	0	0
9	0	0	0	1	0	0	0

A peu près pareil mais sans doute pas de manière significative. Voyons avec un arbre de décision.

[13]:
```
from sklearn.tree import DecisionTreeClassifier
clr = DecisionTreeClassifier()
clr.fit(X_train, y_train)
```

[13]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

[14]:
```
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[14]:
```
59.50769230769231
```

Et avec OneVsRestClassifier :

[15]:
```
clr = OneVsRestClassifier(DecisionTreeClassifier())
clr.fit(X_train, y_train)
```

[15]:
```
OneVsRestClassifier(estimator=DecisionTreeClassifier(class_weight=None,
criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          n_jobs=1)
```

[16]:
```
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[16]:
```
52.92307692307693
```

Et avec OneVsOneClassifier

[17]:
```
clr = OneVsOneClassifier(DecisionTreeClassifier())
clr.fit(X_train, y_train)
```

[17]:
```
OneVsOneClassifier(estimator=DecisionTreeClassifier(class_weight=None,
criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          n_jobs=1)
```

[18]:
```
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[18]:
```
60.12307692307692
```

Mieux.

[19]:
```
from sklearn.ensemble import RandomForestClassifier
clr = RandomForestClassifier()
clr.fit(X_train, y_train)
```

[19]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

[20]:
```
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[20]:
```
66.46153846153847
```

[21]:
```
clr = OneVsRestClassifier(RandomForestClassifier())
clr.fit(X_train, y_train)
```

[21]:
```
OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
          n_jobs=1)
```

[22]:
```
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[22]:
```
65.90769230769232
```

Proche, il faut affiner avec une validation croisée.

[23]:
```
from sklearn.neural_network import MLPClassifier
clr = MLPClassifier(hidden_layer_sizes=30, max_iter=600)
clr.fit(X_train, y_train)
```

[23]:
```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=30, learning_rate='constant',
       learning_rate_init=0.001, max_iter=600, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

[24]:
```
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[24]:
```
51.323076923076925
```

[25]:
```
clr = OneVsRestClassifier(MLPClassifier(hidden_layer_sizes=30, max_iter=600))
clr.fit(X_train, y_train)
```

[25]:
```
OneVsRestClassifier(estimator=MLPClassifier(activation='relu', alpha=0.0001,
batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=30, learning_rate='constant',
       learning_rate_init=0.001, max_iter=600, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False),
          n_jobs=1)
```

[26]:
```
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[26]:
```
47.56923076923077
```

Pas foudroyant.

[27]:
```

```