wines_multi_stacking

November 26, 2021

1 Classification multi-classe et stacking

On cherche à prédire la note d'un vin avec un classifieur multi-classe puis à améliorer le score obtenu avec une méthode dite de stacking.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

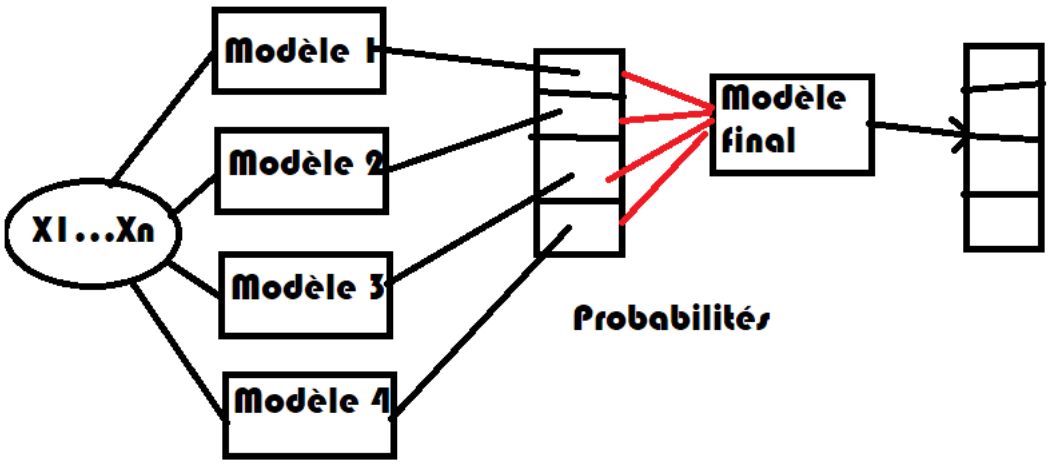
[1]:
```
<IPython.core.display.HTML object>
```

1.1 Le problème

Il n'est pas évident que les scores des différents modèles qu'on apprend sur chacun des classes soient comparables. Si le modèle n'est pas assez performant, on peut songer à ajouter un dernier modèle qui prend la décision finale en fonction du résultat de chaque modèle.

[2]:
```
from pyquickhelper.helpgen import NbImage
NbImage('images/stackmulti.png', width=400)
```

[2]:



[3]:
```
%matplotlib inline
```

[4]:
```
from papierstat.datasets import load_wines_dataset
df = load_wines_dataset()
```

```
X = df.drop(['quality', 'color'], axis=1)
y = df['quality']
```

[5]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

[6]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
clr = OneVsRestClassifier(LogisticRegression(max_iter=1500))
clr.fit(X_train, y_train)
```

[6]:
```
OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None,
                                                 dual=False, fit_intercept=True,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, max_iter=1500,
                                                 multi_class='auto',
                                                 n_jobs=None, penalty='l2',
                                                 random_state=None,
                                                 solver='lbfgs', tol=0.0001,
                                                 verbose=0, warm_start=False),
                    n_jobs=None)
```

[7]:
```
import numpy
numpy.mean(clr.predict(X_test).ravel() == y_test.ravel()) * 100
```

[7]:
```
53.907692307692315
```

On regarde la matrice de confusion.

[8]:
```
from sklearn.metrics import confusion_matrix
import pandas
df = pandas.DataFrame(confusion_matrix(y_test, clr.predict(X_test)))
try:
    df.columns = [str(_) for _ in clr.classes_][:df.shape[1]]
    df.index = [str(_) for _ in clr.classes_][:df.shape[0]]
except ValueError:
    # Il peut arriver qu'une classe ne soit pas représenter
    # lors de l'apprentissage
    print("erreur", df.shape, clr.classes_)
df
```

[8]:

	3	4	5	6	7	8	9
3	0	0	2	1	0	1	0
4	0	0	39	15	1	0	0
5	0	0	314	201	2	0	0
6	0	0	170	537	9	0	0
7	0	0	17	233	25	0	0
8	0	0	2	47	6	0	0
9	0	0	0	2	1	0	0

On cale d'abord une random forest sur les données brutes.

[9]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
```

```
numpy.mean(rfc.predict(X_test).ravel() == y_test.ravel()) * 100
```

[9]:
```
67.44615384615385
```

On cale une random forest avec les sorties de la régression logistique.

[10]:
```
rf_train = clr.decision_function(X_train)

rfc_y = RandomForestClassifier()
rfc_y.fit(rf_train, y_train)
```

[10]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

On calcule le taux d'erreur.

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

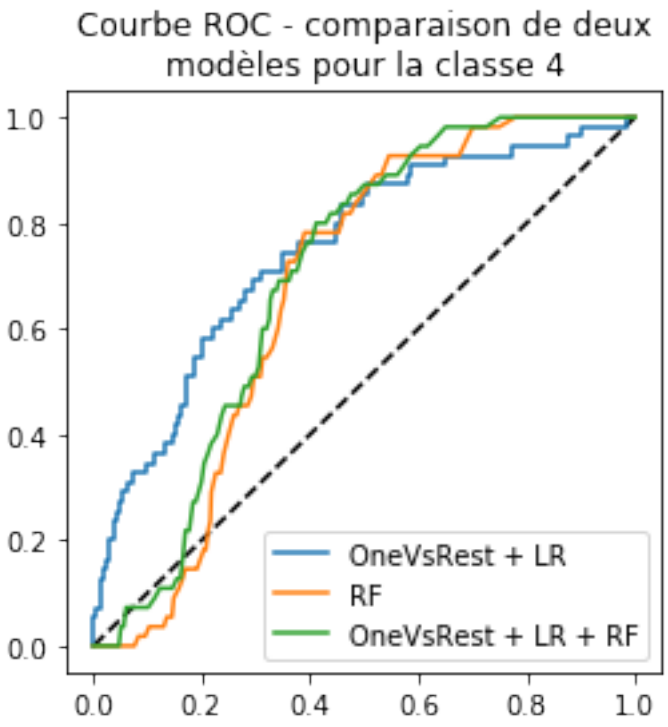
[11]:
```
64.8
```

C'est presque équivalent à une random forest calée sur les données brutes. On trace les courbes ROC pour la classe 4.

[12]:
```
from sklearn.metrics import roc_curve, roc_auc_score
fpr_lr, tpr_lr, th_lr = roc_curve(y_test == 4, clr.decision_function(X_test)[:, 2])
fpr_rfc, tpr_rfc, th_rfc = roc_curve(y_test == 4, rfc.predict_proba(X_test)[:, 2])
fpr_rfc_y, tpr_rfc_y, th_rfc_y = roc_curve(y_test == 4, rfc_y.predict_proba(rf_test)[:
↪, 2])
auc_lr = roc_auc_score(y_test == 4, clr.decision_function(X_test)[:, 2])
auc_rfc = roc_auc_score(y_test == 4, rfc.predict_proba(X_test)[:, 2])
auc_rfc_y = roc_auc_score(y_test == 4, rfc_y.predict_proba(rf_test)[:, 2])
auc_lr, auc_rfc, auc_rfc_y
```

[12]:
```
(0.7485466126230457, 0.6752634626519977, 0.6984597568037059)
```

[13]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(4,4))
ax.plot([0, 1], [0, 1], 'k--')
ax.plot(fpr_lr, tpr_lr, label="OneVsRest + LR")
ax.plot(fpr_rfc, tpr_rfc, label="RF")
ax.plot(fpr_rfc_y, tpr_rfc_y, label="OneVsRest + LR + RF")
ax.set_title('Courbe ROC - comparaison de deux\nmodèles pour la classe 4')
ax.legend();
```



La courbe ROC ne montre rien de probant. Il faudrait vérifier avec une cross-validation qu'il serait pratique de faire avec un pipeline mais ceux-ci n'acceptent qu'un seul prédicteur final.

[14]:
```
from sklearn.pipeline import make_pipeline
try:
    pipe = make_pipeline(OneVsRestClassifier(LogisticRegression(max_iter=1500)),
                         RandomForestClassifier())
except Exception as e:
    print('ERREUR :')
    print(e)
```

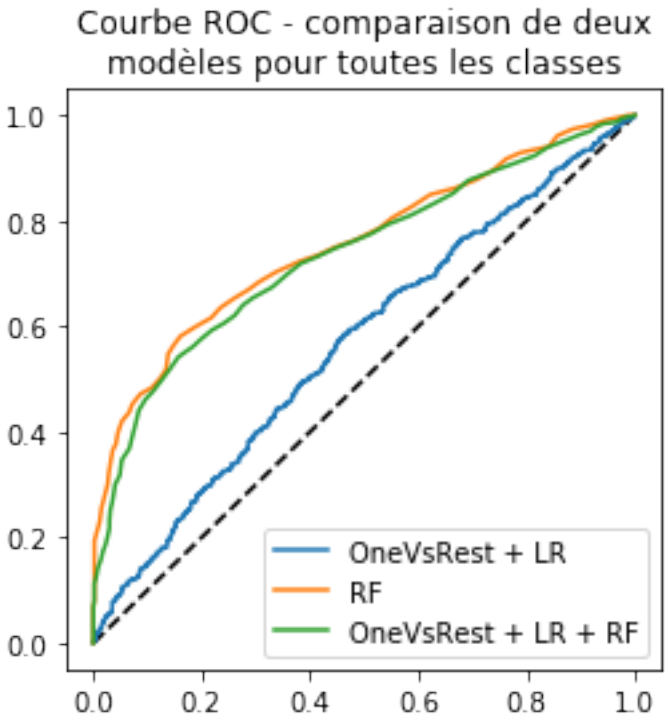
```
ERREUR :
All intermediate steps should be transformers and implement fit and transform or
be the string 'passthrough'
'OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None,
                                                  dual=False, fit_intercept=True,
                                                  intercept_scaling=1,
                                                  l1_ratio=None, max_iter=1500,
                                                  multi_class='auto',
                                                  n_jobs=None, penalty='l2',
                                                  random_state=None,
                                                  solver='lbfgs', tol=0.0001,
                                                  verbose=0, warm_start=False),
                    n_jobs=None)' (type <class
'sklearn.multiclass.OneVsRestClassifier'>) doesn't
```

On construit une ROC sur toutes les classes.

[15]:
```
fpr_lr, tpr_lr, th_lr = roc_curve(y_test == clr.predict(X_test),
                                  clr.predict_proba(X_test).max(axis=1),␣
 ↪drop_intermediate=False)
fpr_rfc, tpr_rfc, th_rfc = roc_curve(y_test == rfc.predict(X_test),
                                  rfc.predict_proba(X_test).max(axis=1),␣
 ↪drop_intermediate=False)
fpr_rfc_y, tpr_rfc_y, th_rfc_y = roc_curve(y_test == rfc_y.predict(rf_test),
                                     rfc_y.predict_proba(rf_test).max(axis=1),␣
 ↪drop_intermediate=False)
auc_lr = roc_auc_score(y_test == clr.predict(X_test),
                       clr.decision_function(X_test).max(axis=1))
auc_rfc = roc_auc_score(y_test == rfc.predict(X_test),
                        rfc.predict_proba(X_test).max(axis=1))
auc_rfc_y = roc_auc_score(y_test == rfc_y.predict(rf_test),
                          rfc_y.predict_proba(rf_test).max(axis=1))
auc_lr, auc_rfc, auc_rfc_y
```

[15]:
```
(0.5510406569489914, 0.753562533633215, 0.7354245943989535)
```

[16]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(4,4))
ax.plot([0, 1], [0, 1], 'k--')
ax.plot(fpr_lr, tpr_lr, label="OneVsRest + LR")
ax.plot(fpr_rfc, tpr_rfc, label="RF")
ax.plot(fpr_rfc_y, tpr_rfc_y, label="OneVsRest + LR + RF")
ax.set_title('Courbe ROC - comparaison de deux\nmodèles pour toutes les classes')
ax.legend();
```



Sur ce modèle, le score produit par le classifieur final paraît plus partinent que le score obtenu en prenant le score maximum sur toutes les classes. On tente une dernière approche où le modèle final doit valider ou non la réponse : c'est un classifieur binaire. Avec celui-ci, tous les classifieurs estimés sont binaires.

[17]:
```
rf_train_bin = clr.decision_function(X_train)
y_train_bin = clr.predict(X_train) == y_train
rfc = RandomForestClassifier()
rfc.fit(rf_train_bin, y_train_bin)
```

[17]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

On regarde les premières réponses.

[18]:
```
rf_test_bin = clr.decision_function(X_test)
rfc.predict_proba(rf_test_bin)[:3]
```

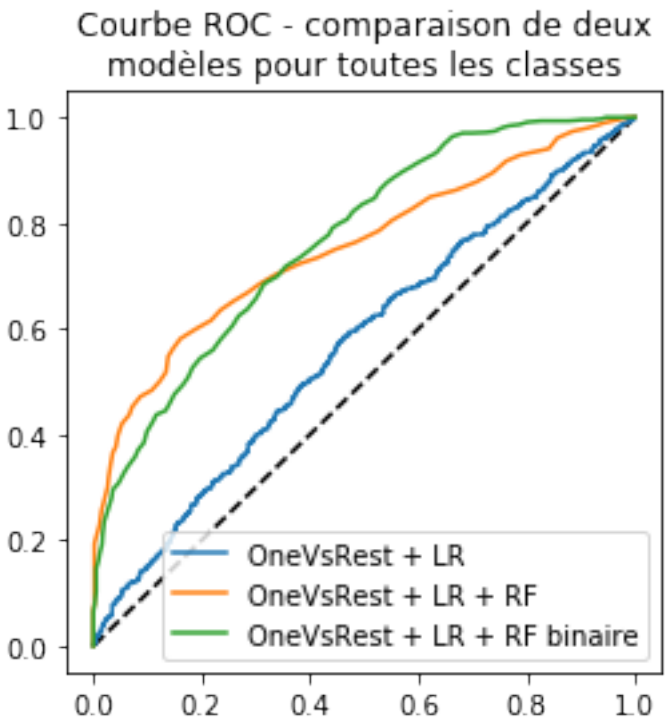
[18]:
```
array([[0.32, 0.68],
       [0.5 , 0.5 ],
       [0.89, 0.11]])
```

[19]:
```
y_test_bin = clr.predict(X_test) == y_test
```

[20]:
```
fpr_rfc_bin, tpr_rfc_bin, th_rfc_bin = roc_curve(y_test_bin, rfc.
 ↪predict_proba(rf_test_bin)[:, 1])
auc_rfc_bin = roc_auc_score(y_test_bin, rfc.predict_proba(rf_test_bin)[:, 1])
auc_lr, auc_rfc_bin
```

[20]:
```
(0.5510406569489914, 0.7668718108162481)
```

[21]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(4,4))
ax.plot([0, 1], [0, 1], 'k--')
ax.plot(fpr_lr, tpr_lr, label="OneVsRest + LR")
ax.plot(fpr_rfc, tpr_rfc, label="OneVsRest + LR + RF")
ax.plot(fpr_rfc_bin, tpr_rfc_bin, label="OneVsRest + LR + RF binaire")
ax.set_title('Courbe ROC - comparaison de deux\nmodèles pour toutes les classes')
ax.legend();
```



Un peu mieux mais il faudrait encore valider avec une validation croisée et plusieurs jeux de données, y compris artificiels. Il reste néanmoins l'idée.

1.2 Automatisation avec une implémentation

Comme c'est fastidieux de faire tout cela, on implémente une classe qui convertit un modèle de machine learning en un *transform* qu'on peut insérer dans un pipeline.

[22]:
```
from mlinsights.sklapi import SkBaseTransformStacking
model = make_pipeline(
                SkBaseTransformStacking(
                    [OneVsRestClassifier(LogisticRegression(max_iter=1500))],
                    'decision_function'),
                RandomForestClassifier())
model.fit(X_train, y_train)
```

[22]:
```
Pipeline(memory=None,
         steps=[('skbasetransformstacking',
SkBaseTransformStacking([OneVsRestClassifier(estimator=LogisticRegress
    ion(C=1.0, class_weight=None,
    dual=False, fit_intercept=True,
    intercept_scaling=1,
    l1_ratio=None, max_iter=1500,
    multi_class='auto',
    n_jobs=None, penalty='l2',
    random_state=None,
    solver='lbfgs', tol=0.0001,
    verbose=0, warm_start=False),                      n_j…
```

```
                    RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                           class_weight=None, criterion='gini',
                                           max_depth=None, max_features='auto',
                                           max_leaf_nodes=None, max_samples=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=100, n_jobs=None,
                                           oob_score=False, random_state=None,
                                           verbose=0, warm_start=False))],
            verbose=False)
```

[23]:
```
fpr_pipe, tpr_pipe, th_pipe = roc_curve(y_test == model.predict(X_test),
                                        model.predict_proba(X_test).max(axis=1),
                                        drop_intermediate=False)
auc_pipe = roc_auc_score(y_test == model.predict(X_test),
                         model.predict_proba(X_test).max(axis=1))
auc_pipe
```

[23]:
```
0.7330188804547779
```

On n'oublie pas de mélanger les données avant de faire tourner la validation croisée.

[24]:
```
df = load_wines_dataset(shuffle=True)
X = df.drop(['quality', 'color'], axis=1)
y = df['quality']
```

On retrouve les mêmes résultats mais on peut maintenant faire une validation croisée.

[25]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
cross_val_score(model, X, y, cv=5)
```

[25]:
```
array([0.66923077, 0.66153846, 0.67513472, 0.66897614, 0.65588915])
```

1.3 scikit-learn 0.22 - stacking

A partir de la version 0.22, *scikit-learn* a introduit le modèle StackingClassifier avec un design un peu différent

[26]:
```
try:
    from sklearn.ensemble import StackingClassifier
    skl = True
except ImportError:
    # scikit-learn pas assez récent
    skl = False
if skl:
    model = StackingClassifier([
        ('ovrlr', OneVsRestClassifier(LogisticRegression(max_iter=1500))),
        ('rf', RandomForestClassifier())
    ])
    model.fit(X_train, y_train)
else:
    model = None
model
```

```
C:\xavierdupre\__home_\github_fork\scikit-
learn\sklearn\model_selection\_split.py:667: UserWarning: The least populated
class in y has only 2 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
C:\xavierdupre\__home_\github_fork\scikit-
learn\sklearn\model_selection\_split.py:667: UserWarning: The least populated
class in y has only 2 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
C:\xavierdupre\__home_\github_fork\scikit-
learn\sklearn\linear_model\_logistic.py:939: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html.
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

[26]:
```
StackingClassifier(cv=None,
                   estimators=[('ovrlr',
OneVsRestClassifier(estimator=LogisticRegression(C=1.0,
 class_weight=None,
 dual=False,
 fit_intercept=True,
 intercept_scaling=1,
 l1_ratio=None,
 max_iter=1500,
 multi_class='auto',
 n_jobs=None,
 penalty='l2',
 random_state=None,
 solver='lbfgs',
 tol=0.0001,
 verbose=0,
 warm_start=False),
                                                  n_jobs=None)),
                               ('rf',
                                RandomForestCla…
                                                     max_depth=None,
                                                     max_features='auto',
                                                     max_leaf_nodes=None,
                                                     max_samples=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=100,
                                                     n_jobs=None,
                                                     oob_score=False,
                                                     random_state=None,
```

```
                                                verbose=0,
                                                warm_start=False))],
                    final_estimator=None, n_jobs=None, passthrough=False,
                    stack_method='auto', verbose=0)
```

[27]:
```
if model is not None:
    fpr_pipe, tpr_pipe, th_pipe = roc_curve(y_test == model.predict(X_test),
                                            model.predict_proba(X_test).max(axis=1),
                                            drop_intermediate=False)
    auc_pipe = roc_auc_score(y_test == model.predict(X_test),
                             model.predict_proba(X_test).max(axis=1))
else:
    auc_pipe = None
auc_pipe
```

[27]:
```
0.7301920159931777
```

[28]:
```

```