wines_color

June 30, 2023

1 Régression logistique et courbe ROC

Prédire la couleur d'un vin à partir de ses composants et visualiser la performance avec une courbe ROC.

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

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

[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)
```

```
C:\xavierdupre\__home_\github_fork\scikit-
learn\sklearn\linear_model\_logistic.py:763: 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
  n_iter_i = _check_optimize_result(
```

[4]:
```
LogisticRegression()
```

La première façon de vérifier que le modèle a marché consiste à regarder la matrice de confusion.

[5]:
```
from sklearn.metrics import confusion_matrix
conf = confusion_matrix(y_test, clr.predict(X_test))
conf
```

[5]:
```
array([[ 397,   22],
       [  10, 1196]], dtype=int64)
```

Les coefficients sur la diagonale indique les éléments bien classés, les coefficients en dehors de ceux que le classifieur a mis dans la mauvaise classe.

[6]:
```
import pandas
cf = pandas.DataFrame(conf, columns=['prédit ' + _ for _ in clr.classes_])
cf.index = ['vrai ' + _ for _ in clr.classes_]
cf
```

[6]:

	prédit red	prédit white
vrai red	397	22
vrai white	10	1196

Un classifieur construit une frontière entre deux classes, la distance d'un point à la frontière constitue une information importante. Plus elle est grande, plus le modèle est confiant. Cette distance est souvent appelée *score*.

[7]:
```
clr.decision_function(X_test)
```

[7]:
```
array([ 8.42164043,  3.4396193 ,  7.67211348, …, -2.83309973,
       -7.57390854,  4.52740831])
```

Mais on préfère les probabilités quand elles sont disponibles :

[8]:
```
clr.predict_proba(X_test)
```

[8]:
```
array([[2.20004961e-04, 9.99779995e-01],
       [3.10799464e-02, 9.68920054e-01],
       [4.65415966e-04, 9.99534584e-01],
       …,
       [9.44438483e-01, 5.55615169e-02],
       [9.99486583e-01, 5.13417044e-04],
       [1.06930748e-02, 9.89306925e-01]])
```

Voyons comment le score est distribué :

[9]:
```
score = clr.decision_function(X_test)
dfsc = pandas.DataFrame(score, columns=['score'])
dfsc['color'] = y_test
dfsc.head()
```

[9]:

	score	color
0	8.421640	NaN
1	3.439619	red
2	7.672113	NaN
3	6.616488	NaN
4	4.080415	NaN

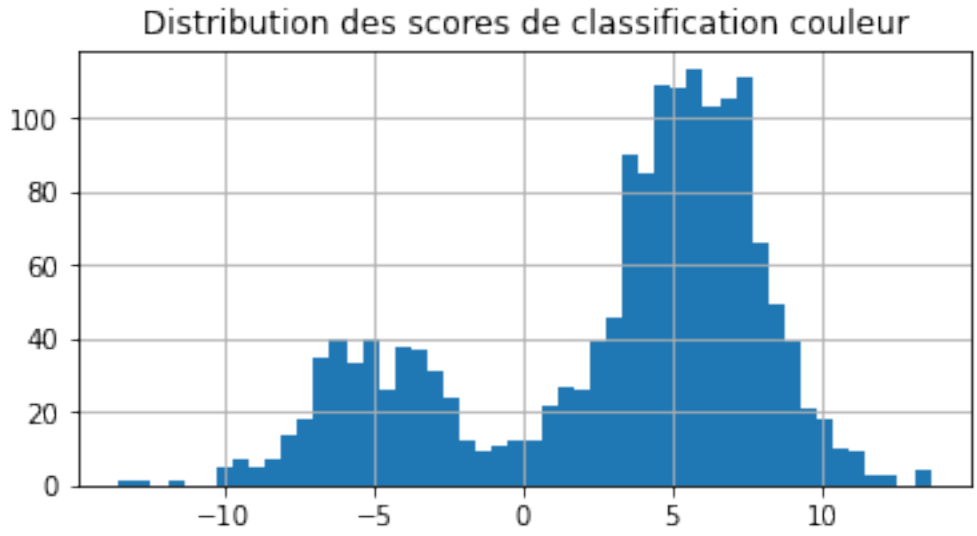
Visiblement, pandas n'a pas compris ce que je voulais qu'il fasse. Il a utilisé les indices de la série *y_test* et a utilisé *y_test.index* comme indice de tableau. Changeons cela.

[10]:
```
dfsc = pandas.DataFrame(score, columns=['score'])
dfsc['color'] = y_test.values
dfsc.head()
```

[10]:

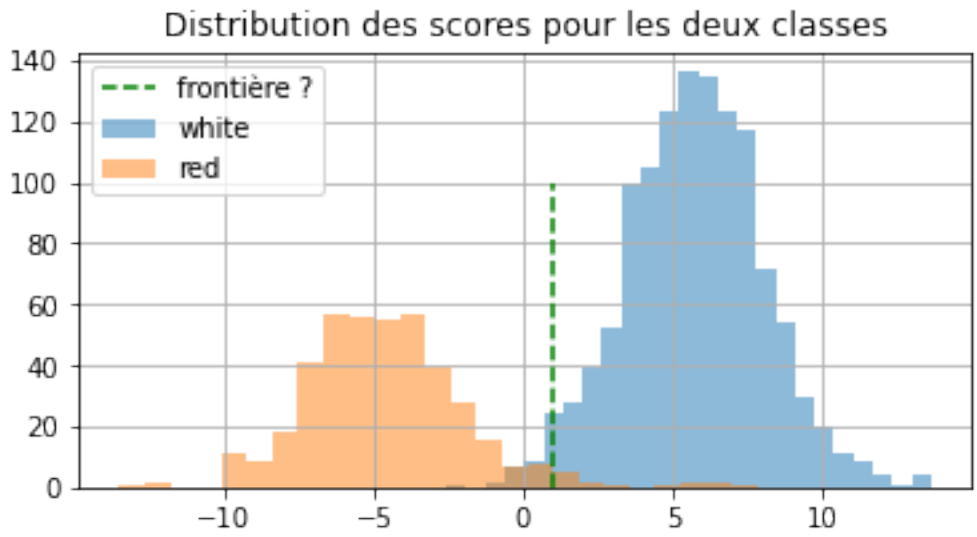
	score	color
0	8.421640	white
1	3.439619	white
2	7.672113	white
3	6.616488	white
4	4.080415	white

[11]:
```
ax = dfsc['score'].hist(bins=50, figsize=(6,3))
ax.set_title('Distribution des scores de classification couleur');
```



Deux modes, probablement les deux classes. Pour en être sûr :

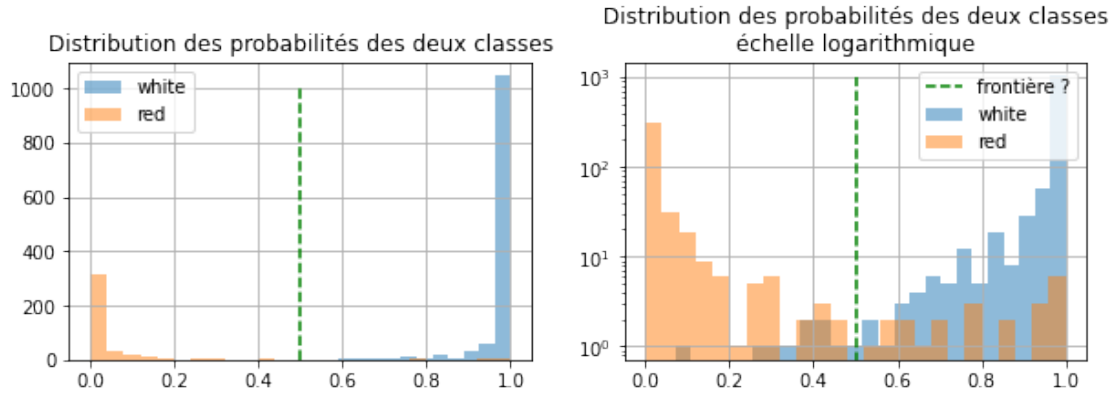
[12]:
```
ax = dfsc[dfsc['color'] == 'white']['score'].hist(bins=25, figsize=(6,3),␣
 ↪label='white', alpha=0.5)
dfsc[dfsc['color'] == 'red']['score'].hist(bins=25, ax=ax, label='red', alpha=0.5)
ax.set_title("Distribution des scores pour les deux classes")
ax.plot([1, 1], [0, 100], 'g--', label="frontière ?")
ax.legend();
```



Il y a quelques confusions autour de 0 mais le modèle est pertinent au sens où la frontière entre les deux classes est assez nette : les deux cloches ne se superposent pas. Voyons avec les probabilités :

[13]:
```
proba = clr.predict_proba(X_test)[:, 1]
dfpr = pandas.DataFrame(proba, columns=['proba'])
dfpr['color'] = y_test.values

import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 2, figsize=(10,3))
dfpr[dfpr['color'] == 'white']['proba'].hist(bins=25, label='white', alpha=0.5,␣
↪ax=ax[0])
dfpr[dfpr['color'] == 'red']['proba'].hist(bins=25, label='red', alpha=0.5, ax=ax[0])
ax[0].set_title('Distribution des probabilités des deux classes')
ax[0].legend();
dfpr[dfpr['color'] == 'white']['proba'].hist(bins=25, label='white', alpha=0.5,␣
↪ax=ax[1])
dfpr[dfpr['color'] == 'red']['proba'].hist(bins=25, label='red', alpha=0.5, ax=ax[1])
ax[0].plot([0.5, 0.5], [0, 1000], 'g--', label="frontière ?")
ax[1].plot([0.5, 0.5], [0, 1000], 'g--', label="frontière ?")
ax[1].set_yscale('log')
ax[1].set_title('Distribution des probabilités des deux classes\néchelle␣
↪logarithmique')
ax[1].legend();
```



Plus l'aire commune aux deux distributions est petite, plus le modèle est confiant. Cette aire commune est reliée à la courbe ROC.

[14]:
```
from sklearn.metrics import roc_auc_score, roc_curve, auc
probas = clr.predict_proba(X_test)
fpr0, tpr0, thresholds0 = roc_curve(y_test, probas[:, 0], pos_label=clr.classes_[0],␣
↪drop_intermediate=False)
fpr0.shape
```

[14]: (1552,)

fpr désigne le False Positive Rate autrement dit le taux de faux positive, si la tâche est déterminer si un vin est blanc, le taux désigne la proportion de vins rouges classés par le classifieur parmi les vins blancs. C'est l'erreur de classification. *tpr* désigne le nombre de True Positive Rate. C'est… A vrai dire, cette dénomination est

toujours aussi absconce pour moi. Je leur préfère les formules mathématiques. On souhaite toujours classer les vins blancs. *True* et *False* ne sont pas vrai ou faux mais le nom de deux classes.

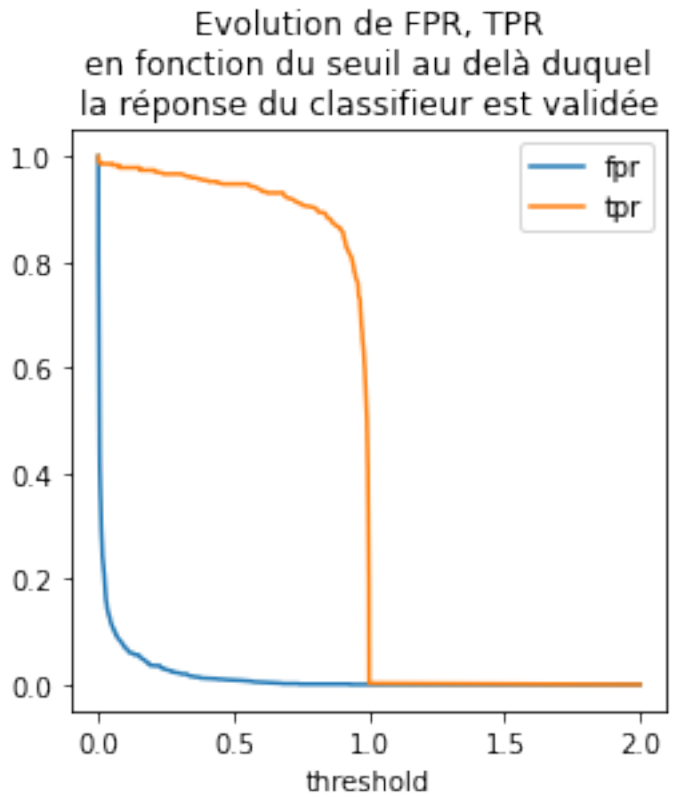
$$\begin{array}{rcl} FPR(s) & = & \sum_{i=1}^{n} \mathbb{1}_{score(X_i) \geqslant s} \mathbb{1}_{y_i == red} \\ TPR(s) & = & \sum_{i=1}^{h} \mathbb{1}_{score(X_i) \geqslant s} \mathbb{1}_{y_i == blanc} \end{array}$$

[15]:
```
dftp = pandas.DataFrame(dict(fpr=fpr0, tpr=tpr0, threshold=thresholds0)).copy()
dftp.head(n=2)
```

[15]:

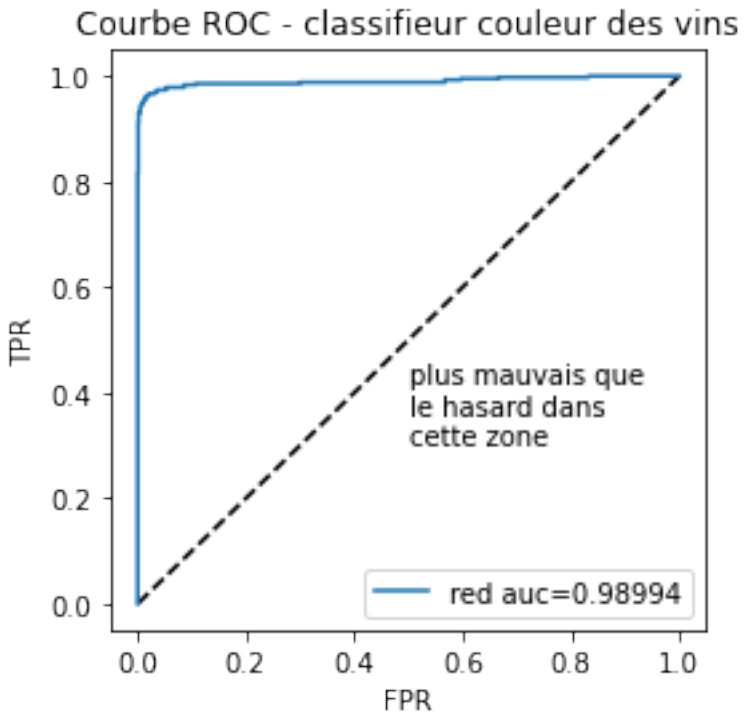
	fpr	tpr	threshold
0	0.0	0.000000	1.999999
1	0.0	0.002387	0.999999

[16]:
```
ax = dftp.plot(x="threshold", y=['fpr', 'tpr'], figsize=(4, 4))
ax.set_title("Evolution de FPR, TPR\nen fonction du seuil au delà duquel\n" +
             "la réponse du classifieur est validée");
```



[17]:
```
fig, ax = plt.subplots(1, 1, figsize=(4,4))
ax.plot([0, 1], [0, 1], 'k--')
# aucf = roc_auc_score(y_test == clr.classes_[0], probas[:, 0]) # première façon
aucf = auc(fpr0, tpr0)  # seconde façon
ax.plot(fpr0, tpr0, label=clr.classes_[0] + ' auc=%1.5f' % aucf)
ax.set_title('Courbe ROC - classifieur couleur des vins')
ax.text(0.5, 0.3, "plus mauvais que\nle hasard dans\ncette zone")
ax.set_xlabel("FPR")
```

```
ax.set_ylabel("TPR");
ax.legend();
```



La mesure AUC ou Area Under the Curve est l'aire sous la courbe. Elle est égale à la probabilité que le score d'un exemple classé rouge à raison soit inférieur à un exemple classé rouge à tort. On vérifie.

[18]:
```
from random import randint
n1, n2 = 0, 0
yt = y_test.values

for n in range(0, 100000):
    i = randint(0, len(yt)-1)
    j = randint(0, len(yt)-1)
    s1, p1 = probas[i, 0], yt[i] == clr.classes_[0]
    s2, p2 = probas[j, 0], yt[j] == clr.classes_[0]
    if p1 != p2:
        if p1:
            if s1 < s2:
                n1 += 1
            else:
                n2 += 1
        else:
            if s1 > s2:
                n1 += 1
            else:
                n2 += 1
print(n2*1.0/(n1 + n2))
```

```
0.990292790429216
```

Presque cela, la fonction auc utilise la fontion trapz et qui calcule une aire et non pas une probabilité comme-ci dessus. Ce théorème qui démontre que cette aire a un lien direct avec les scores de classification. Deux autres métriques sont très utilisées, la précision et le rappel. Pour chaque classifieur, on peut déterminer un seuil s au delà duquel la réponse est validée avec une bonne confiance. Parmi toutes les réponses validées, la précision est le nombre de réponses correctes rapporté au nombre de réponses validées, le rappel est le nombre de réponses correctes rapportées à toutes qui aurait dû être validées. On calcule aussi la métrique *F1* qui est une sorte de moyenne entre les deux.

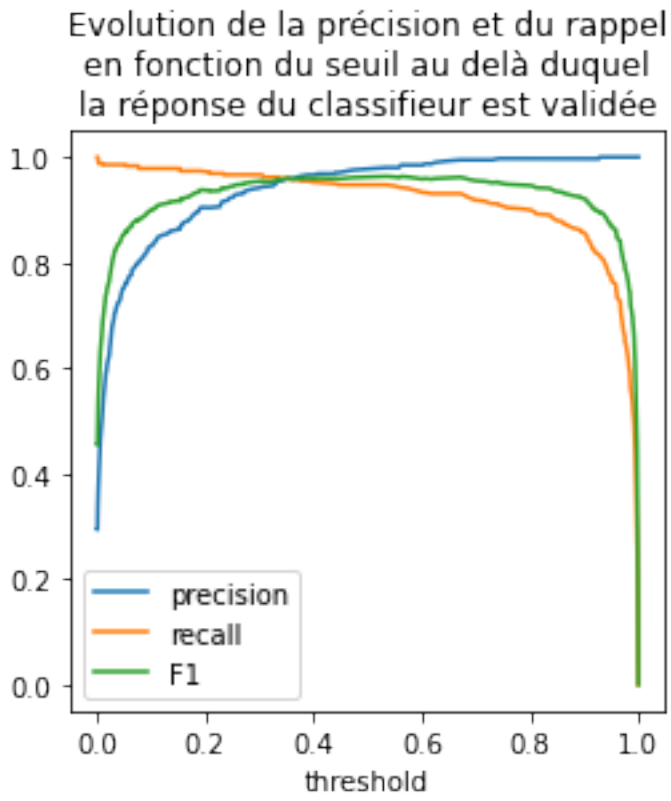
[19]:
```
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(y_test, probas[:, 0],␣
↪pos_label=clr.classes_[0])
```

[20]:
```
dfpr = pandas.DataFrame(dict(precision=precision, recall=recall,
                             threshold=[0] + list(thresholds)))
dfpr['F1']= 2 * (dfpr.precision * dfpr.recall) / (dfpr.precision + dfpr.recall)
dfpr.head(n=2)
```

[20]:

	precision	recall	threshold	F1
0	0.294448	1.000000	0.000000	0.454940
1	0.293952	0.997613	0.000397	0.454101

[21]:
```
ax = dfpr.plot(x="threshold", y=['precision', 'recall', 'F1'], figsize=(4, 4))
ax.set_title("Evolution de la précision et du rappel\nen fonction du seuil au delà␣
↪duquel\n" +
             "la réponse du classifieur est validée");
```



[22]:

```

```