# artificiel_multiclass

November 26, 2021

## 1 Classification multi-classe et jeu mal balancé

Plus il y a de classes, plus la classification est difficile car le nombre d'exemples par classe diminue. Voyons cela plus en détail sur des jeux artificiels produits mar make_blobs.

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

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

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

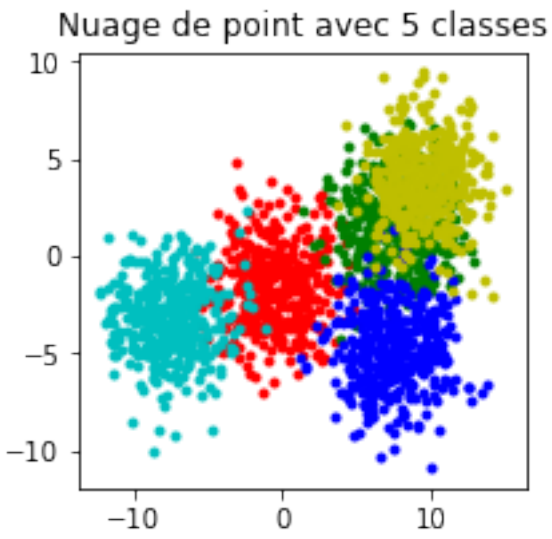
### 1.1 découverte

Le premier jeu de données est une simple fonction linéaire sur deux variables d'ordre de grandeur différents.

[3]:
```
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
X, y = make_blobs(2000, cluster_std=2, centers=5)
```

[4]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(3,3))
for i, c in zip(range(0,5), "rgbyc"):
    ax.plot(X[y==i, 0], X[y==i, 1], c + '.', label=str(i))
ax.set_title("Nuage de point avec 5 classes")
```

[4]:
```
Text(0.5,1,'Nuage de point avec 5 classes')
```




[5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

[6]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
score = model.score(X_test, y_test)
score
```

[6]: 0.8

Mettons le jour dans une fonction pour plusieurs modèles :

[7]:
```
from time import perf_counter as clock

def evaluate_model(models, X_train, X_test, y_train, y_test):
    res = {}
    for k, v in models.items():
        t1 = clock()
        v.fit(X_train, y_train)
        t2 = clock() - t1
        res[k + "_time_train"] = t2
        t1 = clock()
        score = v.score(X_test, y_test)
        t2 = clock() - t1
        res[k + "_time_test"] = t2
        res[k + "_score"] = score
    return res

from sklearn.multiclass import OneVsOneClassifier, OneVsRestClassifier
models = {'OvO-LR': OneVsOneClassifier(LogisticRegression()),
          'OvR-LR': OneVsRestClassifier(LogisticRegression()),
          'LR':  LogisticRegression()}

res = evaluate_model(models, X_train, X_test, y_train, y_test)
```

```
res
```

[7]:
```
{'LR_score': 0.8,
 'LR_time_test': 0.00019950617283950867,
 'LR_time_train': 0.004796839506172837,
 'OvO-LR_score': 0.828,
 'OvO-LR_time_test': 0.001883654320987655,
 'OvO-LR_time_train': 0.017101432098765433,
 'OvR-LR_score': 0.8,
 'OvR-LR_time_test': 0.0004175802469135806,
 'OvR-LR_time_train': 0.011554370370370368}
```

La stratégie *OneVsOne* a l'air d'être plus performante. La régression logistique implémente la stratégie *OneVsRest*. On ne l'évalue plus.

[8]:
```
import pandas

models = {'OvO-LR': OneVsOneClassifier(LogisticRegression()),
          'OvR-LR': LogisticRegression()}

rows = []
for centers in range(2, 51):
    X, y = make_blobs(1000, centers=centers, cluster_std=2.)
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    res = evaluate_model(models, X_train, X_test, y_train, y_test)
    res['centers'] = centers
    rows.append(res)

df = pandas.DataFrame(rows)
df
```

[8]:

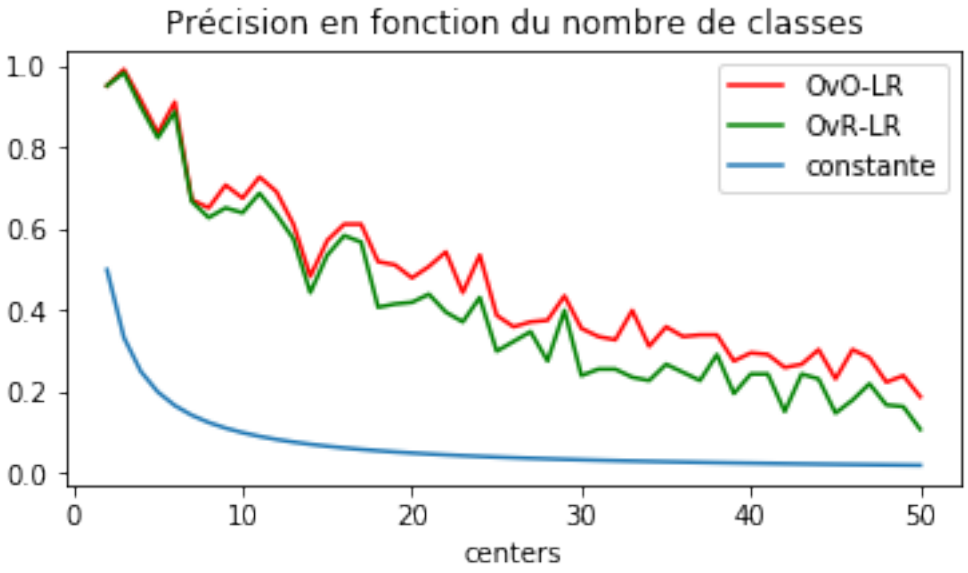
| | OvO-LR_score | OvO-LR_time_test | OvO-LR_time_train | OvR-LR_score | \ |
|---|---|---|---|---|---|
| 0 | 0.952 | 0.000532 | 0.002063 | 0.952 | |
| 1 | 0.992 | 0.000544 | 0.004426 | 0.984 | |
| 2 | 0.916 | 0.000899 | 0.007395 | 0.900 | |
| 3 | 0.836 | 0.001192 | 0.011177 | 0.824 | |
| 4 | 0.912 | 0.001561 | 0.015192 | 0.888 | |
| 5 | 0.672 | 0.002408 | 0.020203 | 0.668 | |
| 6 | 0.652 | 0.002834 | 0.025439 | 0.628 | |
| 7 | 0.708 | 0.003260 | 0.032378 | 0.652 | |
| 8 | 0.676 | 0.003947 | 0.041043 | 0.640 | |
| 9 | 0.728 | 0.006353 | 0.048126 | 0.688 | |
| 10 | 0.692 | 0.007213 | 0.054750 | 0.636 | |
| 11 | 0.612 | 0.006620 | 0.063601 | 0.576 | |
| 12 | 0.484 | 0.007632 | 0.070398 | 0.444 | |
| 13 | 0.572 | 0.008704 | 0.076279 | 0.536 | |
| 14 | 0.612 | 0.009727 | 0.086437 | 0.584 | |
| 15 | 0.612 | 0.012595 | 0.099712 | 0.568 | |
| 16 | 0.520 | 0.015390 | 0.171209 | 0.408 | |
| 17 | 0.512 | 0.014082 | 0.132571 | 0.416 | |
| 18 | 0.480 | 0.018106 | 0.183300 | 0.420 | |
| 19 | 0.508 | 0.016890 | 0.150232 | 0.440 | |
| 20 | 0.544 | 0.019270 | 0.155151 | 0.396 | |
| 21 | 0.444 | 0.041613 | 0.174126 | 0.372 | |

| | | | | |
|---|---|---|---|---|
| 22 | 0.536 | 0.022867 | 0.248510 | 0.432 |
| 23 | 0.388 | 0.024000 | 0.203256 | 0.300 |
| 24 | 0.360 | 0.028209 | 0.272829 | 0.324 |
| 25 | 0.372 | 0.030869 | 0.281628 | 0.348 |
| 26 | 0.376 | 0.033455 | 0.311880 | 0.276 |
| 27 | 0.436 | 0.034357 | 0.274645 | 0.400 |
| 28 | 0.356 | 0.033832 | 0.290394 | 0.240 |
| 29 | 0.336 | 0.041573 | 0.318997 | 0.256 |
| 30 | 0.328 | 0.043851 | 0.386017 | 0.256 |
| 31 | 0.400 | 0.045328 | 0.425602 | 0.236 |
| 32 | 0.312 | 0.050537 | 0.423347 | 0.228 |
| 33 | 0.360 | 0.054385 | 0.528077 | 0.268 |
| 34 | 0.336 | 0.055324 | 0.490342 | 0.248 |
| 35 | 0.340 | 0.054205 | 0.507438 | 0.228 |
| 36 | 0.340 | 0.056871 | 0.454536 | 0.292 |
| 37 | 0.276 | 0.060169 | 0.477994 | 0.196 |
| 38 | 0.296 | 0.063460 | 0.503931 | 0.244 |
| 39 | 0.292 | 0.067772 | 0.530222 | 0.244 |
| 40 | 0.260 | 0.069890 | 0.556523 | 0.152 |
| 41 | 0.268 | 0.071358 | 0.573457 | 0.244 |
| 42 | 0.304 | 0.074807 | 0.591276 | 0.232 |
| 43 | 0.232 | 0.082837 | 0.619394 | 0.148 |
| 44 | 0.304 | 0.084838 | 0.668522 | 0.180 |
| 45 | 0.284 | 0.087833 | 0.693436 | 0.220 |
| 46 | 0.224 | 0.092493 | 0.757865 | 0.168 |
| 47 | 0.240 | 0.092896 | 0.737348 | 0.164 |
| 48 | 0.188 | 0.097161 | 0.762905 | 0.108 |

| | OvR-LR_time_test | OvR-LR_time_train | centers |
|---|---|---|---|
| 0 | 0.000269 | 0.000924 | 2 |
| 1 | 0.002238 | 0.001805 | 3 |
| 2 | 0.000179 | 0.002021 | 4 |
| 3 | 0.000256 | 0.002694 | 5 |
| 4 | 0.000171 | 0.002806 | 6 |
| 5 | 0.000257 | 0.003714 | 7 |
| 6 | 0.000218 | 0.003463 | 8 |
| 7 | 0.000283 | 0.004034 | 9 |
| 8 | 0.000188 | 0.004158 | 10 |
| 9 | 0.000274 | 0.004526 | 11 |
| 10 | 0.000230 | 0.004984 | 12 |
| 11 | 0.000257 | 0.006242 | 13 |
| 12 | 0.000321 | 0.006658 | 14 |
| 13 | 0.000185 | 0.006172 | 15 |
| 14 | 0.000249 | 0.006417 | 16 |
| 15 | 0.000220 | 0.006894 | 17 |
| 16 | 0.000256 | 0.007193 | 18 |
| 17 | 0.000213 | 0.007611 | 19 |
| 18 | 0.000230 | 0.007786 | 20 |
| 19 | 0.000278 | 0.009180 | 21 |
| 20 | 0.000199 | 0.008766 | 22 |
| 21 | 0.000382 | 0.010922 | 23 |
| 22 | 0.000277 | 0.009862 | 24 |
| 23 | 0.000210 | 0.009881 | 25 |

| | | | |
|---|---|---|---|
| 24 | 0.000262 | 0.010441 | 26 |
| 25 | 0.000220 | 0.010645 | 27 |
| 26 | 0.000314 | 0.011112 | 28 |
| 27 | 0.000316 | 0.011742 | 29 |
| 28 | 0.000211 | 0.011475 | 30 |
| 29 | 0.000383 | 0.015168 | 31 |
| 30 | 0.000355 | 0.013080 | 32 |
| 31 | 0.000386 | 0.013297 | 33 |
| 32 | 0.000317 | 0.016676 | 34 |
| 33 | 0.000308 | 0.015390 | 35 |
| 34 | 0.000334 | 0.015256 | 36 |
| 35 | 0.000274 | 0.014890 | 37 |
| 36 | 0.000290 | 0.015210 | 38 |
| 37 | 0.000257 | 0.015356 | 39 |
| 38 | 0.000218 | 0.015635 | 40 |
| 39 | 0.000224 | 0.016497 | 41 |
| 40 | 0.000237 | 0.016856 | 42 |
| 41 | 0.000223 | 0.016344 | 43 |
| 42 | 0.000231 | 0.017263 | 44 |
| 43 | 0.000299 | 0.017989 | 45 |
| 44 | 0.000219 | 0.018234 | 46 |
| 45 | 0.000235 | 0.019005 | 47 |
| 46 | 0.000219 | 0.018962 | 48 |
| 47 | 0.000275 | 0.019491 | 49 |
| 48 | 0.000252 | 0.019164 | 50 |

[9]:
```
fix, ax = plt.subplots(1, 1, figsize=(6, 3))
for c, col in zip('rgb', [_ for _ in df.columns if '_score' in _]):
    df.plot(x="centers", y=col, label=col.replace("_score", ""), ax=ax, color=c)
x = list(range(2, 51))
ax.plot(x, [1./_ for _ in x], label="constante")
ax.legend()
ax.set_title('Précision en fonction du nombre de classes');
```



## 1.2 évolution en fonction du nombre de classes

On pourrait se dire que c'est parce que le nombre d'exemples par classes décroît. Voyons cela.

[10]:
```
import pandas

models = {'OvO-LR': OneVsOneClassifier(LogisticRegression()),
          'OvR-LR': OneVsRestClassifier(LogisticRegression())}

rows = []
for centers in range(2, 51):
    X, y = make_blobs(100 * centers, centers=centers, cluster_std=2.)
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    res = evaluate_model(models, X_train, X_test, y_train, y_test)
    res['centers'] = centers
    rows.append(res)

df2 = pandas.DataFrame(rows)
df2
```

[10]:

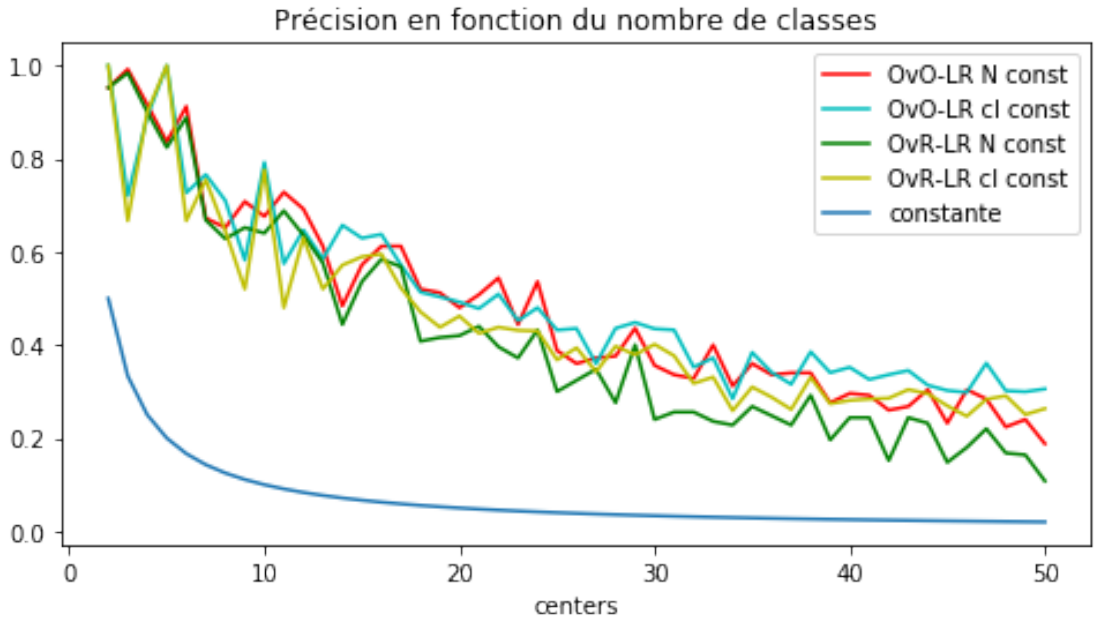
| | OvO-LR_score | OvO-LR_time_test | OvO-LR_time_train | OvR-LR_score | \ |
|---|---|---|---|---|---|
| 0 | 1.000000 | 0.000587 | 0.001734 | 1.000000 | |
| 1 | 0.720000 | 0.001779 | 0.006996 | 0.666667 | |
| 2 | 0.890000 | 0.000717 | 0.010589 | 0.900000 | |
| 3 | 1.000000 | 0.001016 | 0.008864 | 1.000000 | |
| 4 | 0.726667 | 0.001410 | 0.011471 | 0.666667 | |
| 5 | 0.765714 | 0.002008 | 0.015882 | 0.754286 | |
| 6 | 0.710000 | 0.002606 | 0.020889 | 0.645000 | |
| 7 | 0.582222 | 0.003113 | 0.030827 | 0.520000 | |
| 8 | 0.792000 | 0.003928 | 0.033526 | 0.776000 | |
| 9 | 0.574545 | 0.004921 | 0.040922 | 0.480000 | |
| 10 | 0.646667 | 0.005769 | 0.049965 | 0.630000 | |
| 11 | 0.584615 | 0.006990 | 0.058826 | 0.520000 | |
| 12 | 0.657143 | 0.008057 | 0.067661 | 0.571429 | |
| 13 | 0.629333 | 0.009568 | 0.077528 | 0.589333 | |
| 14 | 0.637500 | 0.010995 | 0.095433 | 0.595000 | |
| 15 | 0.571765 | 0.012461 | 0.101595 | 0.522353 | |
| 16 | 0.513333 | 0.014072 | 0.115011 | 0.471111 | |
| 17 | 0.503158 | 0.015938 | 0.131094 | 0.437895 | |
| 18 | 0.492000 | 0.017952 | 0.139939 | 0.462000 | |
| 19 | 0.478095 | 0.020930 | 0.153735 | 0.424762 | |
| 20 | 0.509091 | 0.023613 | 0.172529 | 0.438182 | |
| 21 | 0.452174 | 0.026570 | 0.188020 | 0.431304 | |
| 22 | 0.480000 | 0.028776 | 0.205142 | 0.430000 | |
| 23 | 0.432000 | 0.032036 | 0.221892 | 0.368000 | |
| 24 | 0.435385 | 0.034969 | 0.241384 | 0.393846 | |
| 25 | 0.360000 | 0.038825 | 0.259441 | 0.342222 | |
| 26 | 0.435714 | 0.043034 | 0.279804 | 0.397143 | |
| 27 | 0.448276 | 0.046252 | 0.304205 | 0.379310 | |
| 28 | 0.434667 | 0.049811 | 0.323254 | 0.401333 | |
| 29 | 0.432258 | 0.053886 | 0.345351 | 0.376774 | |

| | | | | |
|---|---|---|---|---|
| 30 | 0.352500 | 0.059383 | 0.367927 | 0.317500 |
| 31 | 0.372121 | 0.063115 | 0.392152 | 0.330909 |
| 32 | 0.284706 | 0.068855 | 0.414858 | 0.258824 |
| 33 | 0.384000 | 0.073336 | 0.443769 | 0.309714 |
| 34 | 0.341111 | 0.079197 | 0.467665 | 0.287778 |
| 35 | 0.315676 | 0.085606 | 0.494793 | 0.261622 |
| 36 | 0.385263 | 0.090279 | 0.524547 | 0.330526 |
| 37 | 0.340513 | 0.096959 | 0.551573 | 0.273846 |
| 38 | 0.352000 | 0.098522 | 0.578178 | 0.281000 |
| 39 | 0.325854 | 0.113506 | 0.636888 | 0.283902 |
| 40 | 0.336190 | 0.125784 | 0.696500 | 0.285714 |
| 41 | 0.345116 | 0.126222 | 0.770164 | 0.304186 |
| 42 | 0.313636 | 0.145638 | 0.804599 | 0.295455 |
| 43 | 0.302222 | 0.143376 | 0.789731 | 0.269333 |
| 44 | 0.298261 | 0.149031 | 0.855741 | 0.246957 |
| 45 | 0.360851 | 0.160348 | 0.893616 | 0.283404 |
| 46 | 0.301667 | 0.168928 | 0.874258 | 0.290833 |
| 47 | 0.299592 | 0.174086 | 0.957626 | 0.250612 |
| 48 | 0.305600 | 0.180501 | 0.891576 | 0.263200 |

| | OvR-LR_time_test | OvR-LR_time_train | centers |
|---|---|---|---|
| 0 | 0.001030 | 0.002107 | 2 |
| 1 | 0.000423 | 0.006687 | 3 |
| 2 | 0.000289 | 0.004599 | 4 |
| 3 | 0.000318 | 0.005234 | 5 |
| 4 | 0.000407 | 0.006223 | 6 |
| 5 | 0.000393 | 0.007808 | 7 |
| 6 | 0.000424 | 0.008969 | 8 |
| 7 | 0.000460 | 0.011013 | 9 |
| 8 | 0.000491 | 0.012444 | 10 |
| 9 | 0.000605 | 0.014100 | 11 |
| 10 | 0.000578 | 0.015752 | 12 |
| 11 | 0.000650 | 0.017826 | 13 |
| 12 | 0.000675 | 0.019690 | 14 |
| 13 | 0.000719 | 0.022380 | 15 |
| 14 | 0.000751 | 0.024434 | 16 |
| 15 | 0.000798 | 0.026743 | 17 |
| 16 | 0.000854 | 0.028983 | 18 |
| 17 | 0.000853 | 0.031490 | 19 |
| 18 | 0.000891 | 0.034457 | 20 |
| 19 | 0.000991 | 0.037687 | 21 |
| 20 | 0.001032 | 0.040654 | 22 |
| 21 | 0.001081 | 0.043396 | 23 |
| 22 | 0.001097 | 0.045880 | 24 |
| 23 | 0.001133 | 0.050149 | 25 |
| 24 | 0.001165 | 0.053239 | 26 |
| 25 | 0.001211 | 0.057264 | 27 |
| 26 | 0.001369 | 0.060778 | 28 |
| 27 | 0.001284 | 0.064984 | 29 |
| 28 | 0.001351 | 0.067520 | 30 |
| 29 | 0.001429 | 0.073068 | 31 |
| 30 | 0.001494 | 0.076645 | 32 |
| 31 | 0.001545 | 0.080264 | 33 |

| | | | |
|---|---|---|---|
| 32 | 0.001518 | 0.086046 | 34 |
| 33 | 0.001598 | 0.088957 | 35 |
| 34 | 0.001663 | 0.093398 | 36 |
| 35 | 0.001642 | 0.097376 | 37 |
| 36 | 0.001715 | 0.105646 | 38 |
| 37 | 0.001747 | 0.108006 | 39 |
| 38 | 0.001777 | 0.107621 | 40 |
| 39 | 0.002085 | 0.136619 | 41 |
| 40 | 0.002034 | 0.145008 | 42 |
| 41 | 0.001972 | 0.129482 | 43 |
| 42 | 0.002044 | 0.149736 | 44 |
| 43 | 0.002062 | 0.144484 | 45 |
| 44 | 0.002105 | 0.144990 | 46 |
| 45 | 0.002142 | 0.164488 | 47 |
| 46 | 0.002208 | 0.161126 | 48 |
| 47 | 0.002227 | 0.166081 | 49 |
| 48 | 0.002218 | 0.171832 | 50 |

[11]:
```
fix, ax = plt.subplots(1, 1, figsize=(8, 4))
for c1, c2, col in zip('rg', 'cy', [_ for _ in df2.columns if '_score' in _]):
    df.plot(x="centers", y=col, label=col.replace("_score", " N const"), ax=ax,␣
 ↪color=c1)
    df2.plot(x="centers", y=col, label=col.replace("_score", " cl const"), ax=ax,␣
 ↪color=c2)
x = list(range(2, 51))
ax.plot(x, [1./_ for _ in x], label="constante")
ax.legend()
ax.set_title('Précision en fonction du nombre de classes');
```



### 1.3 évolution en fonction de la variance

Un peu mieux mais cela décroît toujours. Peut-être que la courbe dépend de la confusion entre les classes ?

[12]:
```
import pandas

models = {'OvO-LR': OneVsOneClassifier(LogisticRegression()),
          'OvR-LR': OneVsRestClassifier(LogisticRegression())}

rows = []
for std_ in range(5, 31):
    X, y = make_blobs(1000, centers=40, cluster_std=std_/10.)
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    res = evaluate_model(models, X_train, X_test, y_train, y_test)
    res['std'] = std_/10.
    rows.append(res)

df3 = pandas.DataFrame(rows)
df3
```

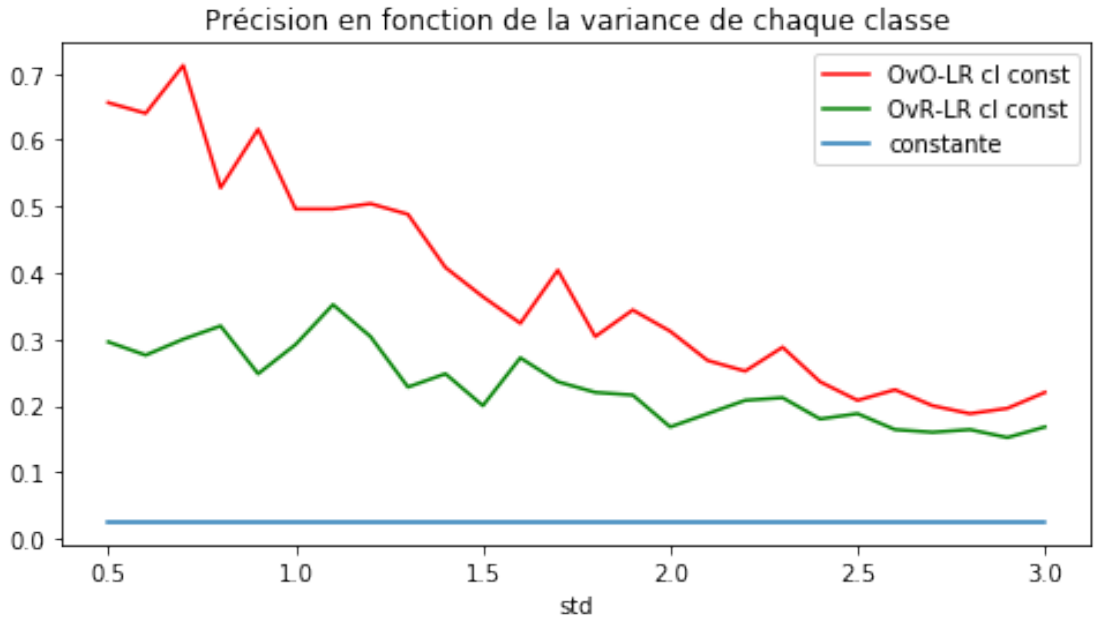
[12]:

| | OvO-LR_score | OvO-LR_time_test | OvO-LR_time_train | OvR-LR_score | \ |
|---|---|---|---|---|---|
| 0 | 0.656 | 0.063483 | 0.539099 | 0.296 | |
| 1 | 0.640 | 0.065259 | 0.533324 | 0.276 | |
| 2 | 0.712 | 0.070078 | 0.524669 | 0.300 | |
| 3 | 0.528 | 0.066854 | 0.567966 | 0.320 | |
| 4 | 0.616 | 0.063002 | 0.505305 | 0.248 | |
| 5 | 0.496 | 0.066295 | 0.534065 | 0.292 | |
| 6 | 0.496 | 0.061540 | 0.512548 | 0.352 | |
| 7 | 0.504 | 0.063630 | 0.503214 | 0.304 | |
| 8 | 0.488 | 0.076549 | 0.552105 | 0.228 | |
| 9 | 0.408 | 0.071063 | 0.602770 | 0.248 | |
| 10 | 0.364 | 0.072232 | 0.579040 | 0.200 | |
| 11 | 0.324 | 0.066767 | 0.513120 | 0.272 | |
| 12 | 0.404 | 0.061820 | 0.522113 | 0.236 | |
| 13 | 0.304 | 0.062575 | 0.529431 | 0.220 | |
| 14 | 0.344 | 0.061557 | 0.510010 | 0.216 | |
| 15 | 0.312 | 0.061025 | 0.487558 | 0.168 | |
| 16 | 0.268 | 0.061219 | 0.489622 | 0.188 | |
| 17 | 0.252 | 0.066159 | 0.513525 | 0.208 | |
| 18 | 0.288 | 0.066587 | 0.554797 | 0.212 | |
| 19 | 0.236 | 0.061922 | 0.520851 | 0.180 | |
| 20 | 0.208 | 0.068296 | 0.509762 | 0.188 | |
| 21 | 0.224 | 0.081953 | 0.640019 | 0.164 | |
| 22 | 0.200 | 0.070516 | 0.623445 | 0.160 | |
| 23 | 0.188 | 0.082961 | 0.546983 | 0.164 | |
| 24 | 0.196 | 0.087073 | 0.860034 | 0.152 | |
| 25 | 0.220 | 0.070357 | 0.567926 | 0.168 | |

| | OvR-LR_time_test | OvR-LR_time_train | std |
|---|---|---|---|
| 0 | 0.001547 | 0.050513 | 0.5 |
| 1 | 0.001362 | 0.051536 | 0.6 |
| 2 | 0.001441 | 0.053237 | 0.7 |
| 3 | 0.001425 | 0.049879 | 0.8 |
| 4 | 0.001404 | 0.048661 | 0.9 |
| 5 | 0.001411 | 0.048941 | 1.0 |

| | | | |
|---|---|---|---|
| 6 | 0.001410 | 0.046235 | 1.1 |
| 7 | 0.001484 | 0.047897 | 1.2 |
| 8 | 0.001436 | 0.056806 | 1.3 |
| 9 | 0.001888 | 0.050014 | 1.4 |
| 10 | 0.001675 | 0.054415 | 1.5 |
| 11 | 0.001708 | 0.051605 | 1.6 |
| 12 | 0.001352 | 0.045884 | 1.7 |
| 13 | 0.001606 | 0.047340 | 1.8 |
| 14 | 0.001356 | 0.045792 | 1.9 |
| 15 | 0.001339 | 0.045506 | 2.0 |
| 16 | 0.001393 | 0.045733 | 2.1 |
| 17 | 0.001530 | 0.049378 | 2.2 |
| 18 | 0.001411 | 0.049761 | 2.3 |
| 19 | 0.001388 | 0.050029 | 2.4 |
| 20 | 0.001392 | 0.046106 | 2.5 |
| 21 | 0.002366 | 0.053813 | 2.6 |
| 22 | 0.002054 | 0.047754 | 2.7 |
| 23 | 0.002633 | 0.084188 | 2.8 |
| 24 | 0.001517 | 0.071725 | 2.9 |
| 25 | 0.001698 | 0.052987 | 3.0 |

[13]:
```
fix, ax = plt.subplots(1, 1, figsize=(8, 4))
for c1, col in zip('rg', [_ for _ in df3.columns if '_score' in _]):
    df3.plot(x="std", y=col, label=col.replace("_score", " cl const"), ax=ax, color=c1)
x = [_/10. for _ in range(5, 31)]
ax.plot(x, [1/40. for _ in x], label="constante")
ax.set_title('Précision en fonction de la variance de chaque classe')
ax.legend();
```



### 1.4 évolution en fonction de la dimension

Et en fonction du nombre de dimensions :

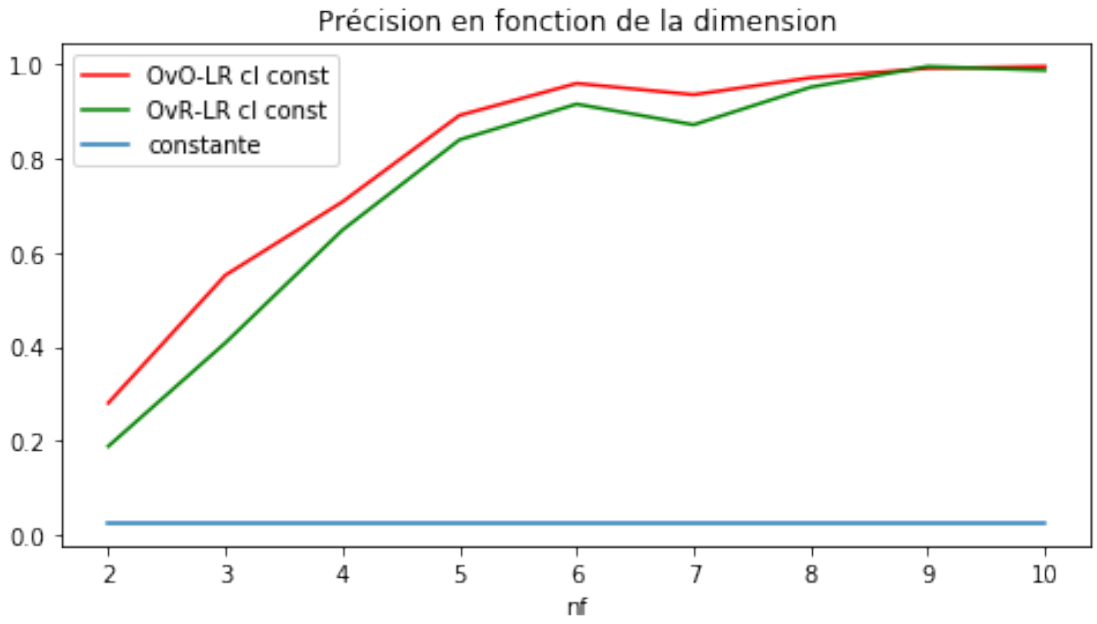
[14]:
```
import pandas

models = {'OvO-LR': OneVsOneClassifier(LogisticRegression()),
          'OvR-LR': OneVsRestClassifier(LogisticRegression())}

rows = []
for nf in range(2, 11):
    X, y = make_blobs(1000, centers=40, cluster_std=2., n_features=nf)
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    res = evaluate_model(models, X_train, X_test, y_train, y_test)
    res['nf'] = nf
    rows.append(res)

df4 = pandas.DataFrame(rows)
df4
```

[14]:

| | OvO-LR_score | OvO-LR_time_test | OvO-LR_time_train | OvR-LR_score | \ |
|---|---|---|---|---|---|
| 0 | 0.280 | 0.072415 | 0.571267 | 0.188 | |
| 1 | 0.552 | 0.077630 | 0.585355 | 0.408 | |
| 2 | 0.708 | 0.084796 | 0.693822 | 0.648 | |
| 3 | 0.892 | 0.092499 | 0.966911 | 0.840 | |
| 4 | 0.960 | 0.088230 | 0.840674 | 0.916 | |
| 5 | 0.936 | 0.091463 | 0.678690 | 0.872 | |
| 6 | 0.972 | 0.125336 | 0.708110 | 0.952 | |
| 7 | 0.992 | 0.073257 | 0.776747 | 0.996 | |
| 8 | 0.996 | 0.071192 | 0.781009 | 0.988 | |

| | OvR-LR_time_test | OvR-LR_time_train | nf |
|---|---|---|---|
| 0 | 0.001486 | 0.054709 | 2 |
| 1 | 0.002046 | 0.058775 | 3 |
| 2 | 0.001385 | 0.062964 | 4 |
| 3 | 0.001447 | 0.079282 | 5 |
| 4 | 0.001685 | 0.086147 | 6 |
| 5 | 0.001480 | 0.086931 | 7 |
| 6 | 0.002739 | 0.176811 | 8 |
| 7 | 0.001535 | 0.106343 | 9 |
| 8 | 0.001511 | 0.115516 | 10 |

[15]:
```
fix, ax = plt.subplots(1, 1, figsize=(8, 4))
for c1, col in zip('rg', [_ for _ in df4.columns if '_score' in _]):
    df4.plot(x="nf", y=col, label=col.replace("_score", " cl const"), ax=ax, color=c1)
x = list(range(2, 11))
ax.plot(x, [1/40. for _ in x], label="constante")
ax.set_title('Précision en fonction de la dimension')
ax.legend();
```




### 1.5 retour sur le nombre de classes

[16]:
```
import pandas

models = {'OvO-LR': OneVsOneClassifier(LogisticRegression()),
          'OvR-LR': OneVsRestClassifier(LogisticRegression())}

rows = []
for centers in range(10, 151, 25):
    X, y = make_blobs(40 * centers, centers=centers, cluster_std=2.)
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    res = evaluate_model(models, X_train, X_test, y_train, y_test)
    res['centers'] = centers
    rows.append(res)

df5 = pandas.DataFrame(rows)
df5
```

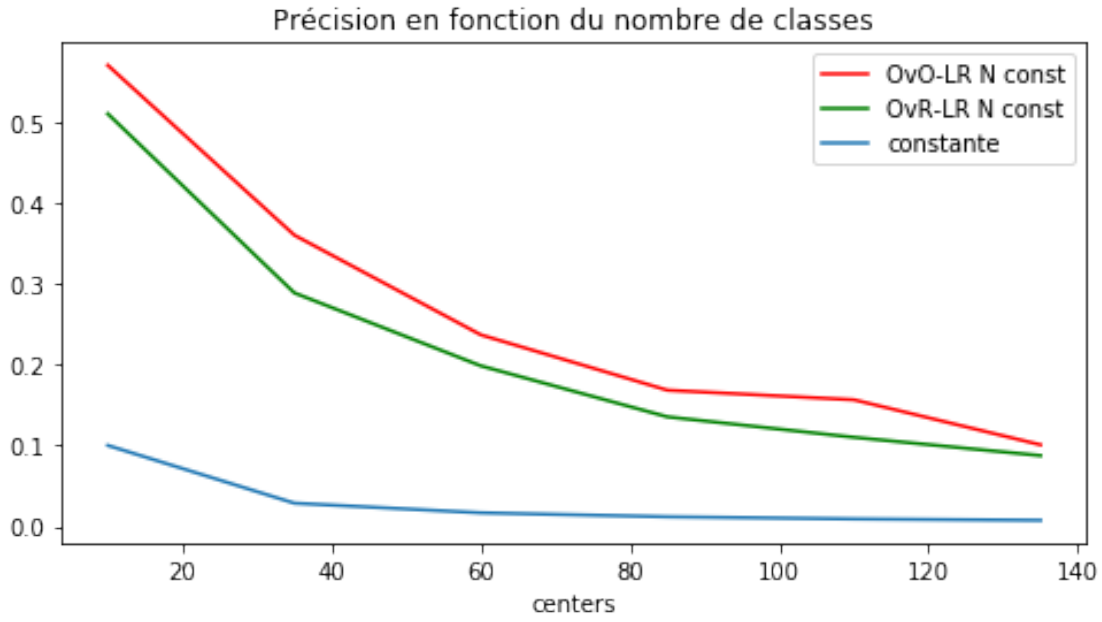
[16]:

| | OvO-LR_score | OvO-LR_time_test | OvO-LR_time_train | OvR-LR_score | \ |
|---|---|---|---|---|---|
| 0 | 0.570000 | 0.005170 | 0.038076 | 0.510000 | |
| 1 | 0.360000 | 0.054092 | 0.513362 | 0.288571 | |
| 2 | 0.236667 | 0.192328 | 1.356538 | 0.198333 | |
| 3 | 0.168235 | 0.559440 | 3.123064 | 0.135294 | |
| 4 | 0.156364 | 1.020811 | 4.499695 | 0.110000 | |
| 5 | 0.100741 | 1.469743 | 6.970833 | 0.087407 | |

| | OvR-LR_time_test | OvR-LR_time_train | centers |
|---|---|---|---|
| 0 | 0.000521 | 0.010232 | 10 |
| 1 | 0.001338 | 0.049240 | 35 |
| 2 | 0.002357 | 0.129873 | 60 |

```
3          0.003981          0.296634        85
4          0.006338          0.467511       110
5          0.005546          0.527650       135
```

[17]:
```
fix, ax = plt.subplots(1, 1, figsize=(8, 4))
for c1, col in zip('rgcy', [_ for _ in df5.columns if '_score' in _]):
    df5.plot(x="centers", y=col, label=col.replace("_score", " N const"), ax=ax,␣
 ↪color=c1)
x = df5.centers
ax.plot(x, [1./_ for _ in x], label="constante")
ax.legend()
ax.set_title('Précision en fonction du nombre de classes');
```




## 1.6 un dernier jeu sûr

On construit un dernier jeu pour lequel le taux de classification devrait être 100%.

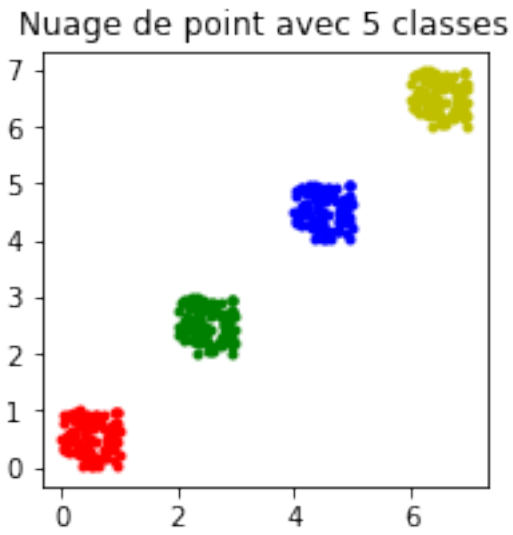
[18]:
```
import numpy

def jeu_x_y(ncl, n):
    uni = numpy.random.random(n*2).reshape((n, 2))
    resx = []
    resy = []
    for i in range(ncl):
        resx.append(uni + i * 2)
        resy.append(numpy.ones(n) * i)
    X = numpy.vstack(resx)
    y = numpy.hstack(resy)
    return X, y
```

```
X, y = jeu_x_y(4, 100)
X.shape, y.shape
```

[18]:
```
((400, 2), (400,))
```

[19]:
```
fig, ax = plt.subplots(1, 1, figsize=(3,3))
for i, c in zip(range(0,5), "rgbyc"):
    ax.plot(X[y==i, 0], X[y==i, 1], c + '.', label=str(i))
ax.set_title("Nuage de point avec 5 classes")
```

[19]:
```
Text(0.5,1,'Nuage de point avec 5 classes')
```




[20]:
```
from sklearn.tree import DecisionTreeClassifier

models = {'OvO-LR': OneVsOneClassifier(LogisticRegression()),
          'OvR-LR': OneVsRestClassifier(LogisticRegression()),
          'DT': DecisionTreeClassifier()}

rows = []
for centers in range(2, 21):
    X, y = jeu_x_y(centers, 10)
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    res = evaluate_model(models, X_train, X_test, y_train, y_test)
    res['centers'] = centers
    rows.append(res)

df5 = pandas.DataFrame(rows)
df5
```

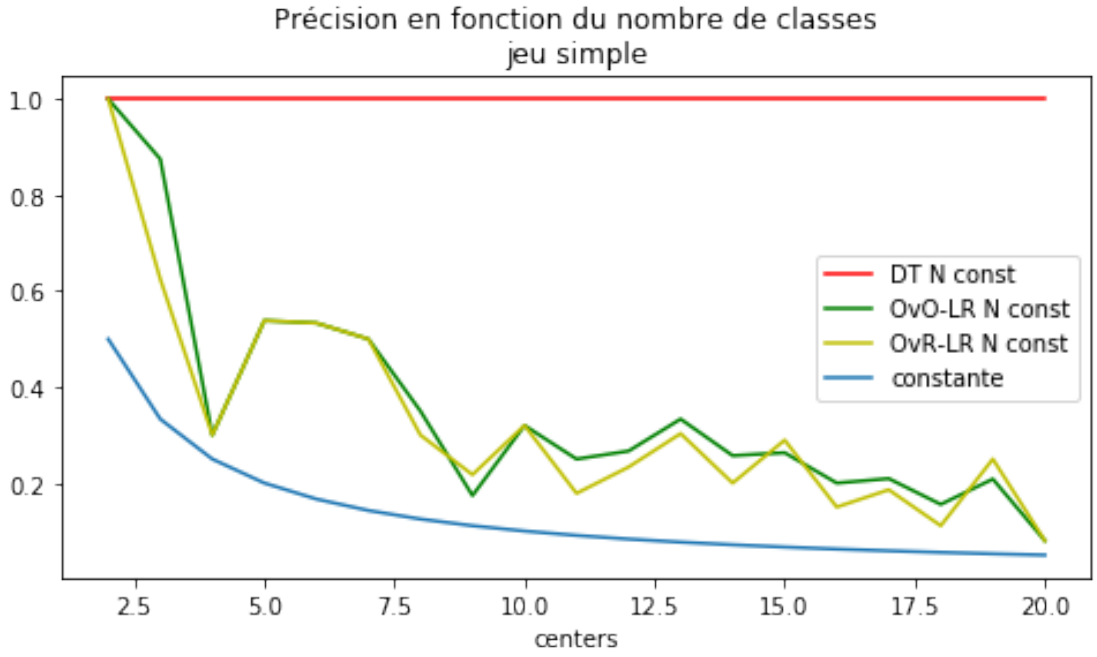
[20]:

| | DT_score | DT_time_test | DT_time_train | OvO-LR_score | OvO-LR_time_test | \ |
|---|---|---|---|---|---|---|
| 0 | 1.0 | 0.000599 | 0.000572 | 1.000000 | 0.000695 | |
| 1 | 1.0 | 0.000213 | 0.000522 | 0.875000 | 0.000773 | |

```
2        1.0      0.000299       0.000359      0.300000          0.000853
3        1.0      0.000176       0.000251      0.538462          0.001852
4        1.0      0.000164       0.000252      0.533333          0.001340
5        1.0      0.000199       0.000274      0.500000          0.001828
6        1.0      0.000167       0.000259      0.350000          0.002699
7        1.0      0.000168       0.000274      0.173913          0.002729
8        1.0      0.000184       0.000321      0.320000          0.003607
9        1.0      0.000173       0.000303      0.250000          0.004306
10       1.0      0.000171       0.000292      0.266667          0.005014
11       1.0      0.000171       0.000305      0.333333          0.005901
12       1.0      0.000177       0.000374      0.257143          0.006479
13       1.0      0.000180       0.000350      0.263158          0.007558
14       1.0      0.000215       0.000388      0.200000          0.009121
15       1.0      0.000179       0.000399      0.209302          0.009530
16       1.0      0.000179       0.000369      0.155556          0.010827
17       1.0      0.000205       0.000492      0.208333          0.019119
18       1.0      0.000185       0.000476      0.080000          0.014104

    OvO-LR_time_train  OvR-LR_score  OvR-LR_time_test  OvR-LR_time_train  \
0            0.001878      1.000000          0.000776           0.002392
1            0.003008      0.625000          0.000434           0.004627
2            0.004449      0.300000          0.000407           0.003547
3            0.015344      0.538462          0.000335           0.007200
4            0.011736      0.533333          0.000418           0.004642
5            0.014497      0.500000          0.000373           0.005886
6            0.018743      0.300000          0.000503           0.006413
7            0.024221      0.217391          0.000443           0.007108
8            0.028823      0.320000          0.000500           0.007797
9            0.036563      0.178571          0.000636           0.008546
10           0.042234      0.233333          0.000514           0.009436
11           0.133461      0.303030          0.000541           0.010604
12           0.063074      0.200000          0.000573           0.011758
13           0.065764      0.289474          0.000606           0.011971
14           0.095704      0.150000          0.000704           0.013614
15           0.091894      0.186047          0.000698           0.013978
16           0.096801      0.111111          0.000684           0.014530
17           0.125401      0.250000          0.001700           0.020256
18           0.126677      0.080000          0.000804           0.017513

    centers
0         2
1         3
2         4
3         5
4         6
5         7
6         8
7         9
8        10
9        11
10       12
11       13
12       14
```

```
13        15
14        16
15        17
16        18
17        19
18        20
```

[21]:
```
fix, ax = plt.subplots(1, 1, figsize=(8, 4))
for c1, col in zip('rgycbp', [_ for _ in df5.columns if '_score' in _]):
    df5.plot(x="centers", y=col, label=col.replace("_score", " N const"), ax=ax,␣
 ↪color=c1)
x = df5.centers
ax.plot(x, [1./_ for _ in x], label="constante")
ax.legend()
ax.set_title('Précision en fonction du nombre de classes\njeu simple');
```




La régression logistique n'est pas le meilleur modèle lorsque le nombre de classes est élevé et la dimension de l'espace de variables faible.

[22]:
```
```