knn_high_dimension

January 1, 2023

1 2A.algo - Plus proches voisins en grande dimension

La méthodes des plus proches voisins est un algorithme assez simple. Que se passe-t-il quand la dimension de l'espace des features augmente ? Comment y remédier ? Le profiling memory_profiler ou cprofile sont des outils utiles pour savoir où le temps est perdu.

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

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

1.1 Q1 : k-nn : mesurer la performance

[2]:
```
from sklearn.datasets import make_classification
datax, datay = make_classification(10000, n_features=10, n_classes=3,
                                   n_clusters_per_class=2, n_informative=8)
datax[:3]
```

[2]:
```
array([[-0.8475772 , -2.32538375,  2.85493495,  0.80844826, -0.22859889,
        -1.04841583,  0.02968567,  0.64623341,  0.80613674, -2.23389406],
       [-0.98432181, -0.06661461,  7.75513731, -0.68528612,  2.91266715,
        -2.42866215, -1.30340144, -2.10535336,  2.30057811, -0.16914582],
       [ 2.5080994 ,  0.78644825,  2.64918709,  1.47316878, -6.35328966,
        -0.82007342, -0.08550633, -5.23436533, -0.56694263, -2.1252314 ]])
```

[3]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(5, algorithm="brute")
model.fit(datax, datay)
```

[3]:
```
KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

[4]:
```
model.predict(datax)
```

[4]:
```
array([2, 1, 0, …, 1, 0, 0])
```

[5]:
```
%timeit model.predict(datax)
```

```
2.78 s ± 27.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[6]:
```
import numpy
import os
path = os.path.normpath(os.path.join(numpy.__file__, '..', '..'))
print(path)
```

```
c:\Python372_x64\lib\site-packages
```

[7]:
```
import cProfile
import pstats
from io import StringIO
pr = cProfile.Profile()
pr.enable()
model.predict(datax)
pr.disable()
s = StringIO()
ps = pstats.Stats(pr, stream=s).sort_stats('cumulative')
ps.print_stats()
res = s.getvalue().replace(path, '').replace("\\", "/").replace(" /", " ")
print('\n'.join(res.split('\n')[:50]))
```

```
         290457 function calls in 2.919 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        2    0.000    0.000    2.918    1.459
IPython/core/interactiveshell.py:3259(run_code)
        2    0.000    0.000    2.918    1.459 {built-in method builtins.exec}
        1    0.000    0.000    2.918    2.918 <ipython-
input-21-b18b3414650f>:6(<module>)
        1    0.000    0.000    2.918    2.918
sklearn/neighbors/classification.py:133(predict)
        1    0.000    0.000    2.600    2.600
sklearn/neighbors/base.py:331(kneighbors)
        2    0.074    0.037    2.599    1.299
sklearn/metrics/pairwise.py:1154(pairwise_distances_chunked)
        1    0.080    0.080    1.277    1.277
sklearn/neighbors/base.py:296(_kneighbors_reduce_func)
        1    0.000    0.000    1.248    1.248
sklearn/metrics/pairwise.py:1315(pairwise_distances)
        1    0.000    0.000    1.248    1.248
sklearn/metrics/pairwise.py:1057(_parallel_pairwise)
        1    0.328    0.328    1.248    1.248
sklearn/metrics/pairwise.py:165(euclidean_distances)
    10003    0.006    0.000    1.210    0.000
numpy/core/fromnumeric.py:54(_wrapfunc)
        1    0.000    0.000    1.197    1.197
numpy/core/fromnumeric.py:742(argpartition)
        1    1.197    1.197    1.197    1.197 {method 'argpartition' of
'numpy.ndarray' objects}
        1    0.000    0.000    0.919    0.919
sklearn/utils/extmath.py:117(safe_sparse_dot)
        1    0.919    0.919    0.919    0.919 {built-in method numpy.dot}
```

```
        1    0.017    0.017    0.318    0.318 scipy/stats/stats.py:389(mode)
    10000    0.016    0.000    0.291    0.000 scipy/stats/stats.py:460(_mode1D)
    10000    0.014    0.000    0.229    0.000
numpy/lib/arraysetops.py:151(unique)
    10000    0.069    0.000    0.204    0.000
numpy/lib/arraysetops.py:299(_unique1d)
    10000    0.052    0.000    0.064    0.000
numpy/lib/function_base.py:1149(diff)
    10000    0.005    0.000    0.037    0.000 {method 'max' of 'numpy.ndarray'
objects}
    10000    0.003    0.000    0.032    0.000 numpy/core/_methods.py:26(_amax)
    10002    0.030    0.000    0.030    0.000 {built-in method
numpy.concatenate}
    10004    0.029    0.000    0.029    0.000 {method 'reduce' of 'numpy.ufunc'
objects}
    30005    0.008    0.000    0.017    0.000
numpy/core/numeric.py:541(asanyarray)
    10000    0.004    0.000    0.017    0.000
numpy/core/fromnumeric.py:1694(nonzero)
    10001    0.007    0.000    0.010    0.000
numpy/lib/index_tricks.py:653(__next__)
    30012    0.009    0.000    0.009    0.000 {built-in method numpy.array}
    10000    0.008    0.000    0.008    0.000 {method 'argmax' of
'numpy.ndarray' objects}
    10002    0.008    0.000    0.008    0.000 {built-in method numpy.empty}
    10000    0.007    0.000    0.007    0.000 {method 'sort' of 'numpy.ndarray'
objects}
    10000    0.007    0.000    0.007    0.000 {method 'flatten' of
'numpy.ndarray' objects}
    10000    0.005    0.000    0.005    0.000 {method 'nonzero' of
'numpy.ndarray' objects}
    10000    0.003    0.000    0.004    0.000
numpy/lib/arraysetops.py:138(_unpack_tuple)
    10000    0.003    0.000    0.003    0.000 {built-in method
numpy.core._multiarray_umath.normalize_axis_index}
    10001    0.003    0.000    0.003    0.000 {built-in method builtins.next}
    20014    0.002    0.000    0.002    0.000 {built-in method builtins.len}
    10012    0.002    0.000    0.002    0.000 {built-in method builtins.getattr}
    10002    0.002    0.000    0.002    0.000 {method 'append' of 'list'
objects}
        3    0.000    0.000    0.001    0.000
sklearn/utils/validation.py:332(check_array)
        3    0.000    0.000    0.001    0.000
sklearn/utils/validation.py:36(_assert_all_finite)
        4    0.000    0.000    0.001    0.000
numpy/core/fromnumeric.py:1966(sum)
        4    0.000    0.000    0.001    0.000
numpy/core/fromnumeric.py:69(_wrapreduction)
        3    0.000    0.000    0.001    0.000
sklearn/utils/extmath.py:663(_safe_accumulator_op)
        1    0.000    0.000    0.000    0.000
numpy/core/fromnumeric.py:942(argsort)
```

Etudier l'évolution du temps de prédiction en fonction du nombre d'observations, de la dimension, du nombre

de classes ? Qu'en déduisez-vous ? Le code sur GitHub :

- predict
- kneighbors
- pairwise_distance

1.2 Q2 : k-nn avec sparse features

On recommence cette mesure de temps mais en créant des jeux de données sparses.

1.3 Q3 : Imaginez une façon d'aller plus vite ?

Aller plus vite veut parfois dire perdre un peu en performance et dans notre cas, on veut accélérer la prédiction.

[8]: