# KNN (100 words by code) : -----------------------------------------------

Fitting 3 folds for each of 3 candidates, totalling 9 fits

train\_r2\_score = 0.8541603355011245

test\_r2\_score = 0.8466541056342308

============================CONFUSION MATRIX=======================================

3. Use SEABORN to draw confusion\_matrix--------------------------------------------------------------

Confusion matrix as graph with Seaborn :

Une image contenant texte, capture d’écran, ligne, Rectangle

Description générée automatiquement

# KNN (100 words by code): -----------------------------------------------

Fitting 3 folds for each of 1 candidates, totalling 3 fits

train\_r2\_score = 0.8645383820579834

test\_r2\_score = 0.8550416337446058

train\_mse\_result = 451240.9234030268

test\_mse\_result = 488600.12933811464

best\_params: [{'n\_neighbors': 2}]

============================CONFUSION MATRIX=======================================

3. Use SEABORN to draw confusion\_matrix--------------------------------------------------------------

Confusion matrix as graph with Seaborn :

Une image contenant texte, capture d’écran, ligne, Rectangle

Description générée automatiquement

# RF (100 words by code): -----------------------------------------------

Fitting 3 folds for each of 1 candidates, totalling 3 fits

train\_r2\_score = 0.8693095484106242

test\_r2\_score = 0.8640977329362426

train\_mse\_result = 446904.01537713484

test\_mse\_result = 465541.4230231569

best\_params: [{'max\_features': 'sqrt', 'min\_samples\_split': 100}]

============================CONFUSION MATRIX=======================================

3. Use SEABORN to draw confusion\_matrix--------------------------------------------------------------

Confusion matrix as graph with Seaborn :

Une image contenant texte, capture d’écran, ligne, Rectangle

Description générée automatiquement

# RF (20 words by code)

estimator RandomForestClassifier()

params {'max\_features': ['sqrt'], 'min\_samples\_split'...

-----------------------------------------------------------------------

Fitting 3 folds for each of 4 candidates, totalling 12 fits

train\_r2\_score = 0.7422506533762839

test\_r2\_score = 0.7427216920926275

============================CONFUSION MATRIX=======================================

3. Use SEABORN to draw confusion\_matrix--------------------------------------------------------------

Confusion matrix as graph with Seaborn :

Une image contenant texte, capture d’écran, ligne, nombre

Description générée automatiquement

train\_f1\_score = [array([0.2868937 , 0.58707865, 0.90558292, 0.94570928, 0.82228999,

0.84216397, 0.7988107 , 0.79822897, 0.8501845 , 0.84218161,

0.88344988, 0.84454176, 0.80547041, 0.93720586, 0.95649241,

0.85385297, 0.92784717, 0.9112426 , 0.30963331, 0.74173927,

0.92497626, 0.74491886, 0.90461875, 0.93111803, 0.86209887,

0.74950242, 0.99928622])]

test\_f1\_score = [array([0.29972752, 0.55950541, 0.89502762, 0.94478528, 0.82327586,

0.84254144, 0.78927203, 0.81287971, 0.87368421, 0.84405797,

0.90566038, 0.85821832, 0.77449168, 0.9369863 , 0.95859649,

0.85185185, 0.92205438, 0.90851735, 0.30957429, 0.74614869,

0.90618762, 0.74152824, 0.90566038, 0.93265633, 0.85 ,

0.75505618, 1. ])]

train\_mse\_result = 379914.35068072693

test\_mse\_result = 378075.26171518874

best\_params: [{'max\_features': 'sqrt', 'min\_samples\_split': 10}]

0

# KNN (20 words by code)

estimator KNeighborsClassifier()

params {'n\_neighbors': [100, 200, 300, 500, 1000]}

-----------------------------------------------------------------------

Fitting 3 folds for each of 5 candidates, totalling 15 fits

train\_r2\_score = 0.7166170303288154

test\_r2\_score = 0.7161611864097733

============================CONFUSION MATRIX=======================================

3. Use SEABORN to draw confusion\_matrix--------------------------------------------------------------

Confusion matrix as graph with Seaborn :

Une image contenant texte, capture d’écran, ligne, nombre

Description générée automatiquement

train\_f1\_score = [array([0.2868937 , 0.58707865, 0.90558292, 0.94570928, 0.82228999,

0.84216397, 0.7988107 , 0.79822897, 0.8501845 , 0.84218161,

0.88344988, 0.84454176, 0.80547041, 0.93720586, 0.95649241,

0.85385297, 0.92784717, 0.9112426 , 0.30963331, 0.74173927,

0.92497626, 0.74491886, 0.90461875, 0.93111803, 0.86209887,

0.74950242, 0.99928622]), array([0.28098032, 0.5795976 , 0.88777639, 0.92307692, 0.82 ,

0.84108671, 0.77575758, 0.29154519, 0.83470456, 0.83898182,

0.85406699, 0.82804569, 0.80037888, 0.93376501, 0.94726097,

0.78611632, 0.92516205, 0.8591674 , 0.55028187, 0.74153239,

0.90310078, 0.74313664, 0.88294314, 0.93067387, 0.84811238,

0.73802009, 0.99928622])]

test\_f1\_score = [array([0.29972752, 0.55950541, 0.89502762, 0.94478528, 0.82327586,

0.84254144, 0.78927203, 0.81287971, 0.87368421, 0.84405797,

0.90566038, 0.85821832, 0.77449168, 0.9369863 , 0.95859649,

0.85185185, 0.92205438, 0.90851735, 0.30957429, 0.74614869,

0.90618762, 0.74152824, 0.90566038, 0.93265633, 0.85 ,

0.75505618, 1. ]), array([0.29041096, 0.55727554, 0.88268156, 0.9148265 , 0.82073434,

0.84013841, 0.75590551, 0.29763899, 0.85866667, 0.83934808,

0.8627451 , 0.83780332, 0.76880223, 0.93347993, 0.95049505,

0.77952756, 0.91944276, 0.85808581, 0.55483871, 0.74614869,

0.88617886, 0.73733333, 0.88335221, 0.93237102, 0.83248731,

0.74090909, 0.99706745])]

train\_mse\_result = 284982.4905184465

test\_mse\_result = 282187.1504892725

best\_params: [{'max\_features': 'sqrt', 'min\_samples\_split': 10}, {'n\_neighbors': 100}]

# SVC (20 words by code)

estimator SVC()

params {'C': [10, 20, 30, 50, 100], 'kernel': ['linea...

-----------------------------------------------------------------------

Fitting 3 folds for each of 15 candidates, totalling 45 fits…

-----------------------------------------------------------------------

# KNN (150 words by code)

estimator KNeighborsClassifier()

params {'n\_neighbors': [10]}

df.shape : (82265, 4052)

X\_train.shape - X\_test.shape - len(y\_train) - len(y\_test)

(65812, 4050) - (16453, 4050) - 65812 - 16453

Fitting 3 folds for each of 1 candidates, totalling 3 fits

[d:\Python312\Lib\site-packages\sklearn\model\_selection\\_search.py:979](file:///D:\Python312\Lib\site-packages\sklearn\model_selection\_search.py:979): UserWarning: One or more of the test scores are non-finite: [nan]

warnings.warn(

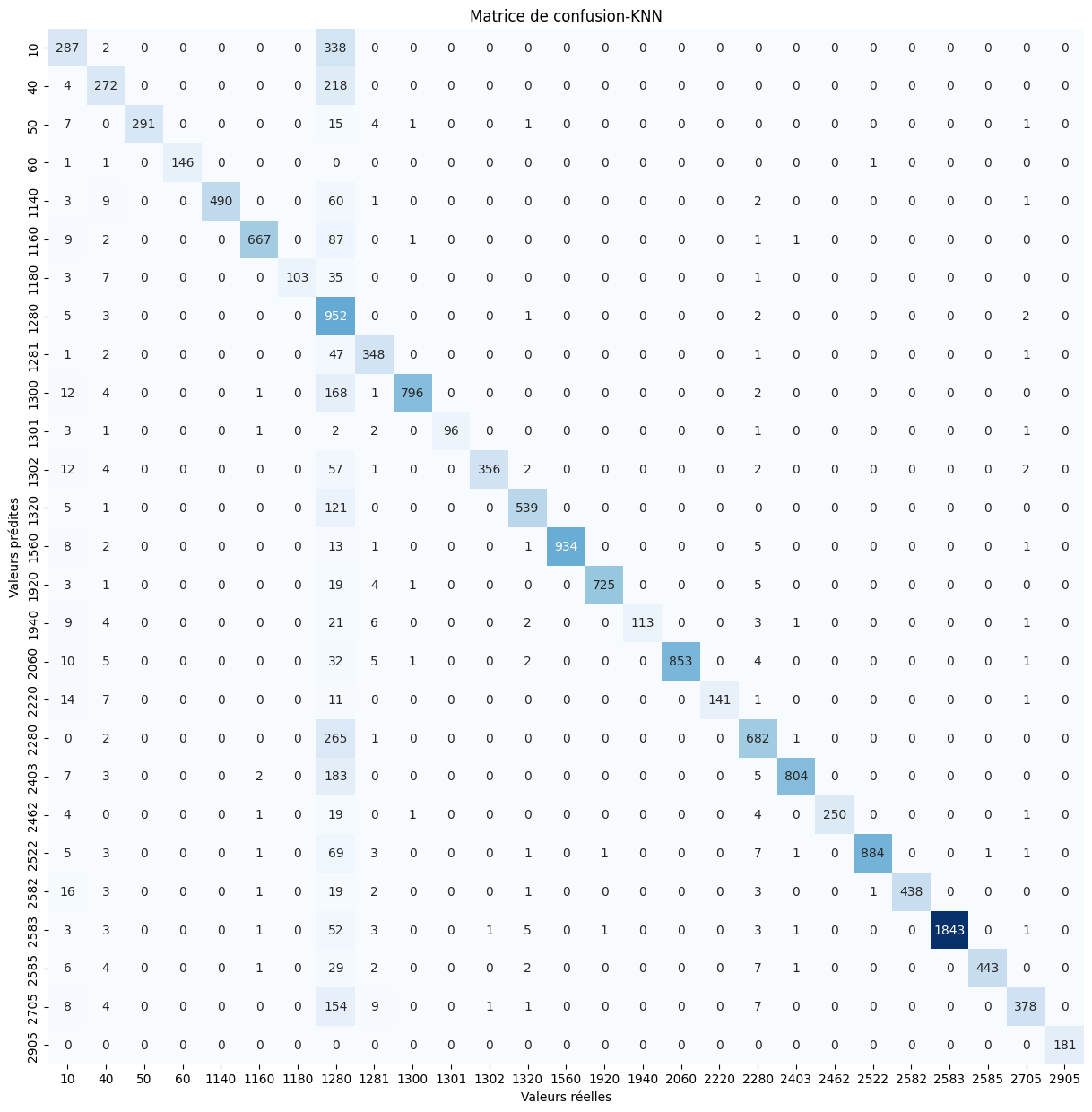
train\_r2\_score = 0.8638242265848174

test\_r2\_score = 0.8516379991490913

============================CONFUSION MATRIX=======================================

3. Use SEABORN to draw confusion\_matrix--------------------------------------------------------------

Confusion matrix as graph with Seaborn :



train\_f1\_score = [array([0.57901204, 0.68607825, 0.95127796, 0.98703404, 0.93035079,

0.93022476, 0.87568556, 0.48681333, 0.88793103, 0.90987821,

0.93099671, 0.92581944, 0.88107058, 0.98251479, 0.98434668,

0.89974293, 0.97252903, 0.94146744, 0.816935 , 0.87853233,

0.95158287, 0.95363889, 0.95509992, 0.98576165, 0.95005429,

0.82464956, 0.99855072])]

test\_f1\_score = [array([0.53544776, 0.64531435, 0.95253682, 0.98983051, 0.9280303 ,

0.92382271, 0.81746032, 0.48190332, 0.8776797 , 0.89187675,

0.94581281, 0.89672544, 0.88071895, 0.98367562, 0.97643098,

0.82783883, 0.96602492, 0.89240506, 0.80282519, 0.88643881,

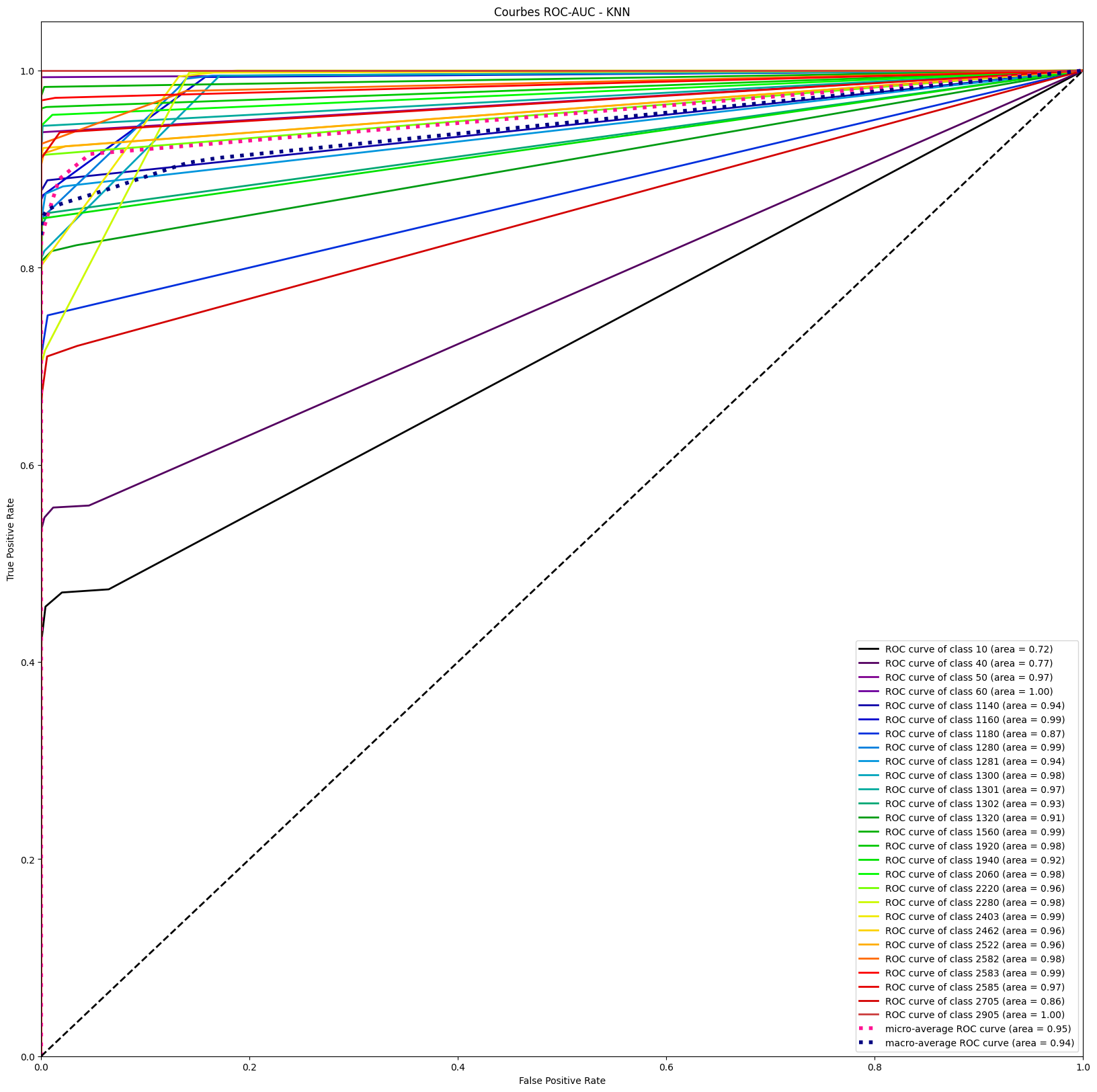
0.94339623, 0.94900698, 0.95010846, 0.98031915, 0.94355698,

0.79162304, 1. ])]

train\_mse\_result = 160215.49750805323

test\_mse\_result = 184986.59539293745

best\_params: [{'n\_neighbors': 10}]



# KNN (150 words by code)

estimator KNeighborsClassifier()

params {'n\_neighbors': [10]}

-----------------------------------------------------------------------

X\_train.shape - X\_test.shape - len(y\_train) - len(y\_test)

(65812, 4050) - (16453, 4050) - 65812 - 16453

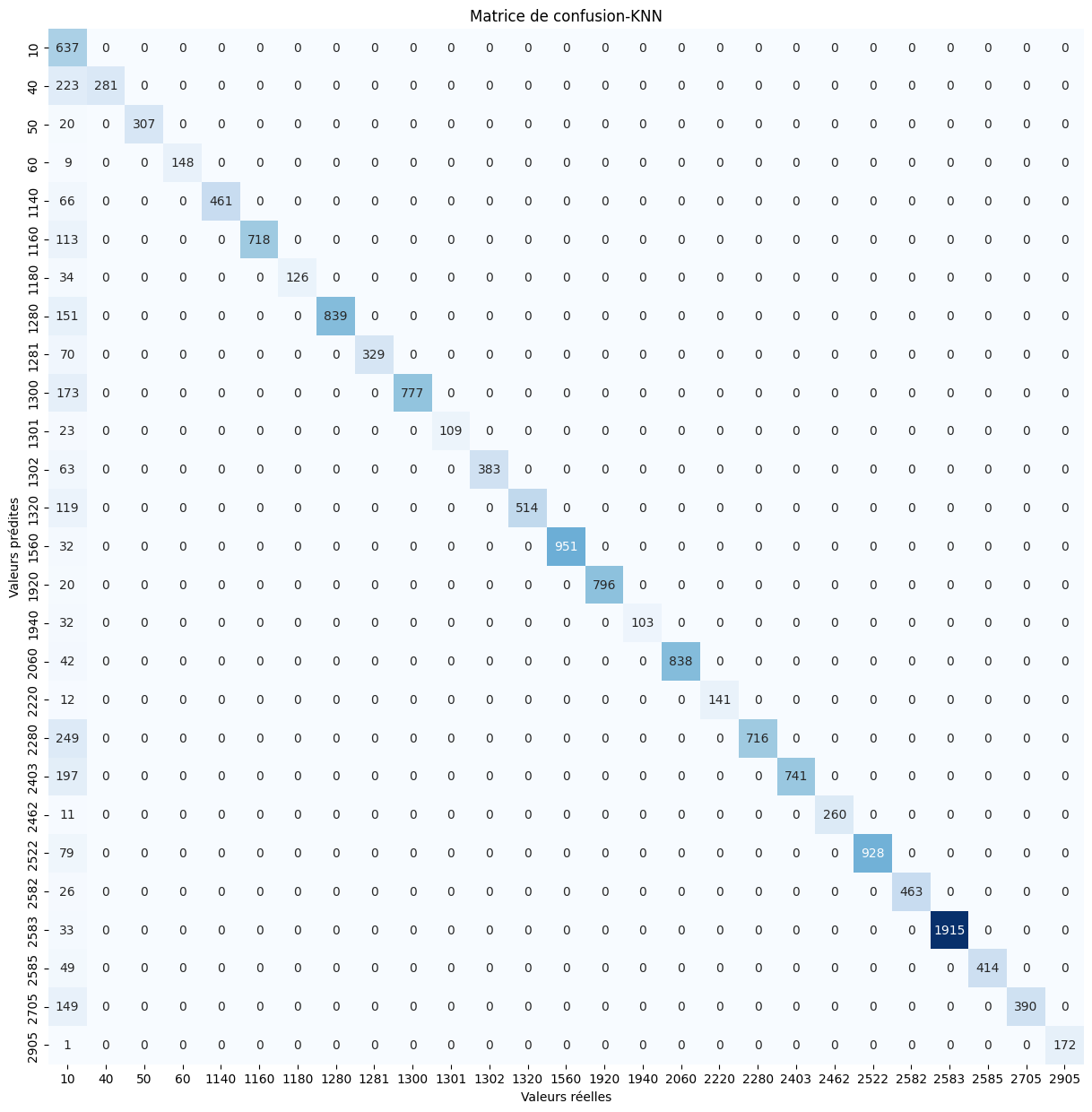
Fitting 3 folds for each of 1 candidates, totalling 3 fits

train\_r2\_score = 0.8857199294961405

test\_r2\_score = 0.8786847383455905

============================CONFUSION MATRIX=======================================

3. Use SEABORN to draw confusion\_matrix--------------------------------------------------------------



train\_f1\_score = [array([0.39711423, 0.72972973, 0.96431404, 0.99018003, 0.94329389,

0.93843537, 0.87940631, 0.9235361 , 0.92734032, 0.91129685,

0.94292237, 0.92778741, 0.89739729, 0.99012947, 0.98742666,

0.92679002, 0.9777964 , 0.95019763, 0.83607313, 0.89386929,

0.97977528, 0.96024384, 0.97210136, 0.98762054, 0.96360759,

0.83718487, 0.99928418])]

test\_f1\_score = [array([0.38960245, 0.71592357, 0.96845426, 0.9704918 , 0.93319838,

0.92704971, 0.88111888, 0.91744122, 0.90384615, 0.89982629,

0.90456432, 0.92400483, 0.89625109, 0.98345398, 0.98759305,

0.86554622, 0.97555297, 0.95918367, 0.85187388, 0.88266825,

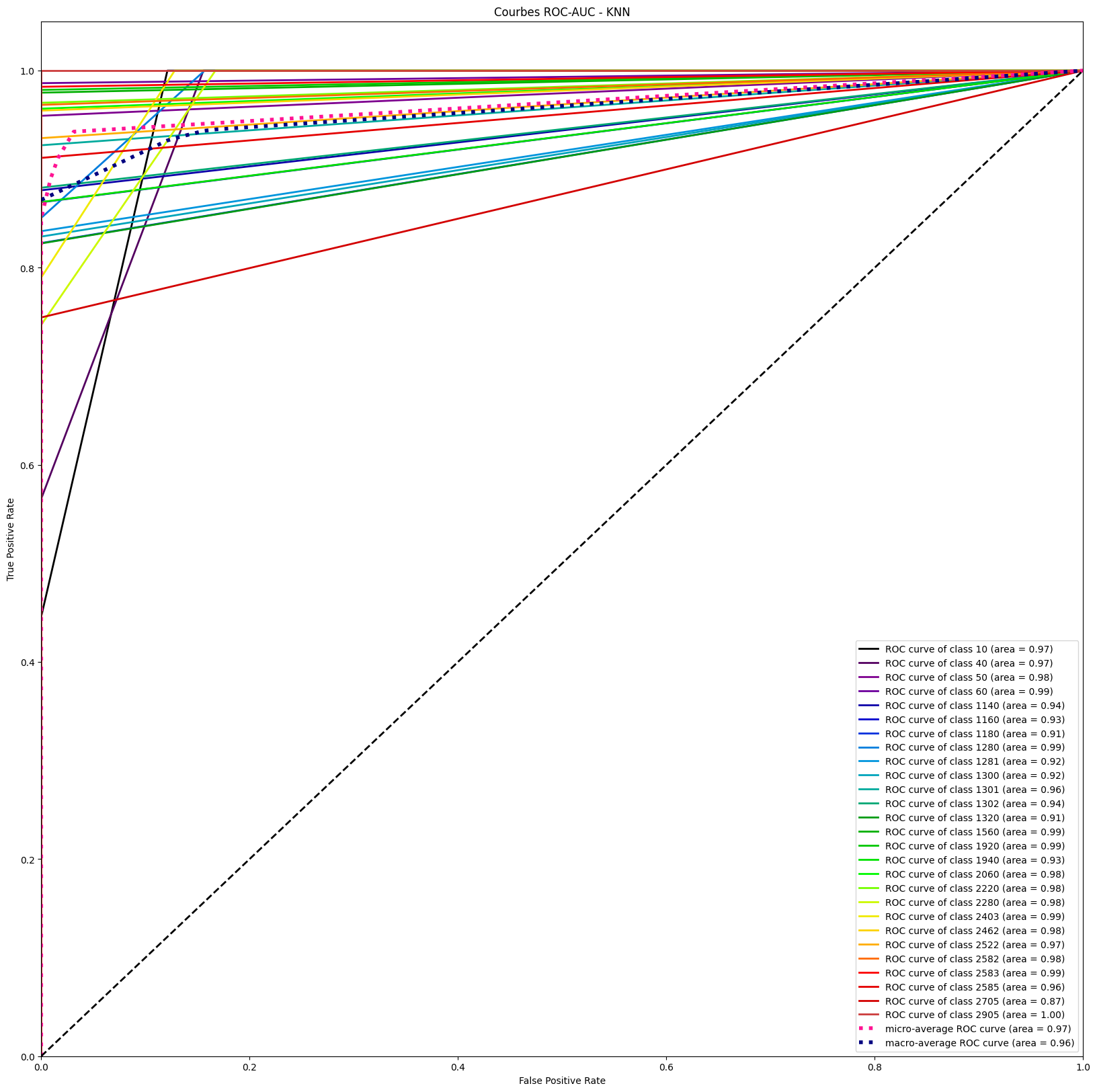
0.97928437, 0.95917313, 0.97268908, 0.99145742, 0.94412771,

0.83961249, 0.99710145])]

train\_mse\_result = 389357.61490305717

test\_mse\_result = 398629.87017565186

best\_params: [{'n\_neighbors': 10}]



-----------------------------------------------------------------------

# KNN (150 words code) avec scaling

estimator KNeighborsClassifier()

params {'n\_neighbors': [10, 12, 30]}

-----------------------------------------------------------------------

df.shape : (82265, 4052)

Fitting 3 folds for each of 3 candidates, totalling 9 fits

**train\_r2\_score = 0.8887436941591199**

**test\_r2\_score = 0.88160213942746**

X\_train BEFORE scaling :

|  | **actes** | **afrique** | **age** | **american** | **ammareal** | **amour** | **analysis** | **ancien** | **annee** | **applications** | **...** | **vidéo\_x.5** | **ville\_x.1** | **vista** | **vive** | **vram** | **windows** | **xp** | **édition\_x.3** | **également\_x.9** | **équipe** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **38528** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **48399** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **3652** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **39161** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **24230** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

5 rows × 4050 columns

X\_train\_scaled (after scaling):

|  | **actes** | **afrique** | **age** | **american** | **ammareal** | **amour** | **analysis** | **ancien** | **annee** | **applications** | **...** | **vidéo\_x.5** | **ville\_x.1** | **vista** | **vive** | **vram** | **windows** | **xp** | **édition\_x.3** | **également\_x.9** | **équipe** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **38528** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |
| **48399** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |
| **3652** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |
| **39161** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |
| **24230** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |

5 rows × 4050 columns

X\_test\_scaled (after scaling):

|  | **actes** | **afrique** | **age** | **american** | **ammareal** | **amour** | **analysis** | **ancien** | **annee** | **applications** | **...** | **vidéo\_x.5** | **ville\_x.1** | **vista** | **vive** | **vram** | **windows** | **xp** | **édition\_x.3** | **également\_x.9** | **équipe** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **49061** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |
| **65875** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |
| **51855** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |
| **18601** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |
| **58740** | -0.014587 | -0.009059 | -0.013085 | -0.014056 | -0.022985 | -0.013644 | -0.012328 | -0.01835 | -0.011695 | -0.009862 | ... | -0.047721 | -0.027734 | -0.049528 | -0.031376 | -0.024973 | -0.070382 | -0.049177 | -0.02932 | -0.035573 | -0.024387 |

5 rows × 4050 columns

X\_train.shape - X\_test.shape - len(y\_train) - len(y\_test)

(65812, 4050) - (16453, 4050) - 65812 - 16453

============================CONFUSION MATRIX=======================================

Use SEABORN to draw confusion\_matrix-----------------------------------------------

Confusion matrix as graph with Seaborn :

Une image contenant texte, capture d’écran, ligne, nombre

Description générée automatiquement

train\_f1\_score = [array([0.41769083, 0.73500967, 0.9689298 , 0.99673736, 0.9419387 ,

0.93805907, 0.90718039, 0.92072124, 0.92792491, 0.91580663,

0.95499451, 0.93367639, 0.83938852, 0.98852649, 0.9897277 ,

0.92972058, 0.97787735, 0.9542903 , 0.83787973, 0.89236564,

0.98163905, 0.95949739, 0.97016461, 0.98893276, 0.96046697,

0.85121825, 0.99854227])]

test\_f1\_score = [array([0.39974043, 0.72592593, 0.97153025, 0.98726115, 0.94706449,

0.93103448, 0.84297521, 0.92807672, 0.92200557, 0.9048928 ,

0.9375 , 0.93544458, 0.83146067, 0.98527171, 0.98066298,

0.8590604 , 0.9728794 , 0.96226415, 0.8453134 , 0.88757396,

0.97328244, 0.96407186, 0.96335079, 0.98697847, 0.950783 ,

0.85623003, 1. ])]

train\_mse\_result = 361571.8485230657

test\_mse\_result = 382027.62973317935

**best\_params: [{'n\_neighbors': 10}]**

# KNN (100 words by code)

X\_train.shape - X\_test.shape - len(y\_train) - len(y\_test)

(67932, 2700) - (16984, 2700) - 67932 - 16984

estimator KNeighborsClassifier()

params {'n\_neighbors': [10], 'weights': ['uniform', '...

-----------------------------------------------------------------------

Fitting 3 folds for each of 2 candidates, totalling 6 fits

**train\_r2\_score = 0.8861802979450039**

**test\_r2\_score = 0.8843028732925106**

**best\_params: [{'algorithm': 'auto', 'n\_jobs': -1, 'n\_neighbors': 10, 'weights': 'distance'}]**

train\_f1\_score = [array([0.39089334, 0.76324655, 0.95494071, 0.99925981, 0.92794814,

0.93105779, 0.8762421 , 0.90616622, 0.91707317, 0.91470786,

0.95813953, 0.94 , 0.90372272, 0.98390572, 0.98402839,

0.95230126, 0.97602475, 0.97179694, 0.81697044, 0.91878173,

0.97751799, 0.95299539, 0.98472906, 0.98477977, 0.959442 ,

0.85405961, 1. ])]

test\_f1\_score = [array([0.39637953, 0.73316062, 0.94256259, 0.99678457, 0.93346981,

0.94455578, 0.92830189, 0.9010503 , 0.87483871, 0.92225201,

0.9453125 , 0.9376392 , 0.90306947, 0.98521698, 0.96850862,

0.94642857, 0.98052921, 0.93103448, 0.81997372, 0.92016083,

0.97472924, 0.94807892, 0.97773475, 0.98227216, 0.9600863 ,

0.8380744 , 0.996997 ])]

train\_mse\_result = 382045.8192162751

test\_mse\_result = 388084.74004945834

============================CONFUSION MATRIX=======================================

Use SEABORN to draw confusion\_matrix-----------------------------------------------

Confusion matrix as graph with Seaborn :

Une image contenant texte, capture d’écran, ligne, Rectangle

Description générée automatiquement

# KNN (300 word by code)

X\_train.shape - X\_test.shape - len(y\_train) - len(y\_test)

**(65812, 8100) - (16453, 8100) - 65812 - 16453**

-----------------------------------------------------------------------

estimator KNeighborsClassifier()

params {'n\_neighbors': [10]}

-----------------------------------------------------------------------

Fitting 3 folds for each of 1 candidates, totalling 3 fits

**train\_r2\_score = 0.9067799185558865**

**test\_r2\_score = 0.9002613505135841**

============================CONFUSION MATRIX=======================================

Use SEABORN to draw confusion\_matrix-----------------------------------------------

Confusion matrix as graph with Seaborn :

train\_f1\_score = [array([0.75349301, 0.8144208 , 0.9837587 , 0.98947368, 0.42818645,

0.96005218, 0.90762332, 0.94754279, 0.95120364, 0.92673847,

0.97002141, 0.95146727, 0.93545683, 0.99200619, 0.99376026,

0.94339623, 0.98420685, 0.964687 , 0.89900759, 0.92226501,

0.98637602, 0.97737438, 0.98398983, 0.99484071, 0.96810207,

0.799908 , 0.97447119])]

test\_f1\_score = [array([0.74541752, 0.8035488 , 0.98245614, 0.97260274, 0.41079812,

0.9569378 , 0.90070922, 0.94072448, 0.95384615, 0.9218573 ,

0.95412844, 0.94033413, 0.92193919, 0.98801199, 0.99282453,

0.88732394, 0.97972973, 0.94153846, 0.91482301, 0.92876563,

0.97707231, 0.97795198, 0.96465696, 0.99503787, 0.96051227,

0.77019749, 0.95031056])]

**train\_mse\_result = 96349.62113292409**

**test\_mse\_result = 103203.23928766791**

**best\_params: [{'n\_neighbors': 10}]**

Une image contenant texte, capture d’écran, ligne, Rectangle

Description générée automatiquement

Une image contenant texte, ligne, capture d’écran, diagramme

Description générée automatiquement