TABLE DES MATIERES

KNN (150 WORDS BY CODE)

TRAIN_R2_SCORE = 0.8638242265848174

TEST_R2_SCORE = 0.8516379991490913

BEST_PARAMS: [{'N_NEIGHBORS': 10}]

estimator KNeighborsClassifier()

params {'n_neighbors': [10]}

---

df.shape : (82265, 4052)

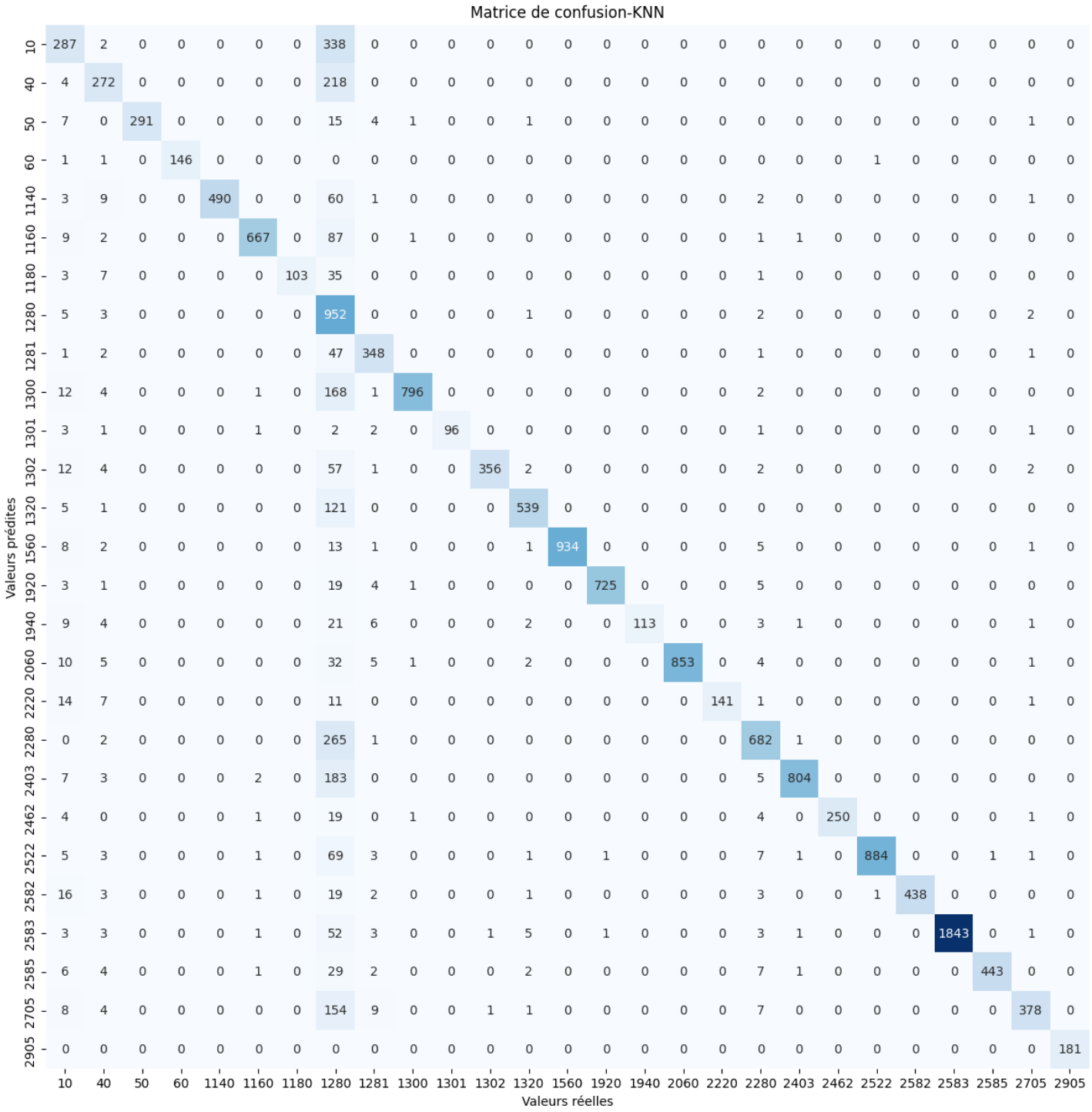
X_train.shape - X_test.shape - len(y_train) - len(y_test)

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

==========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,

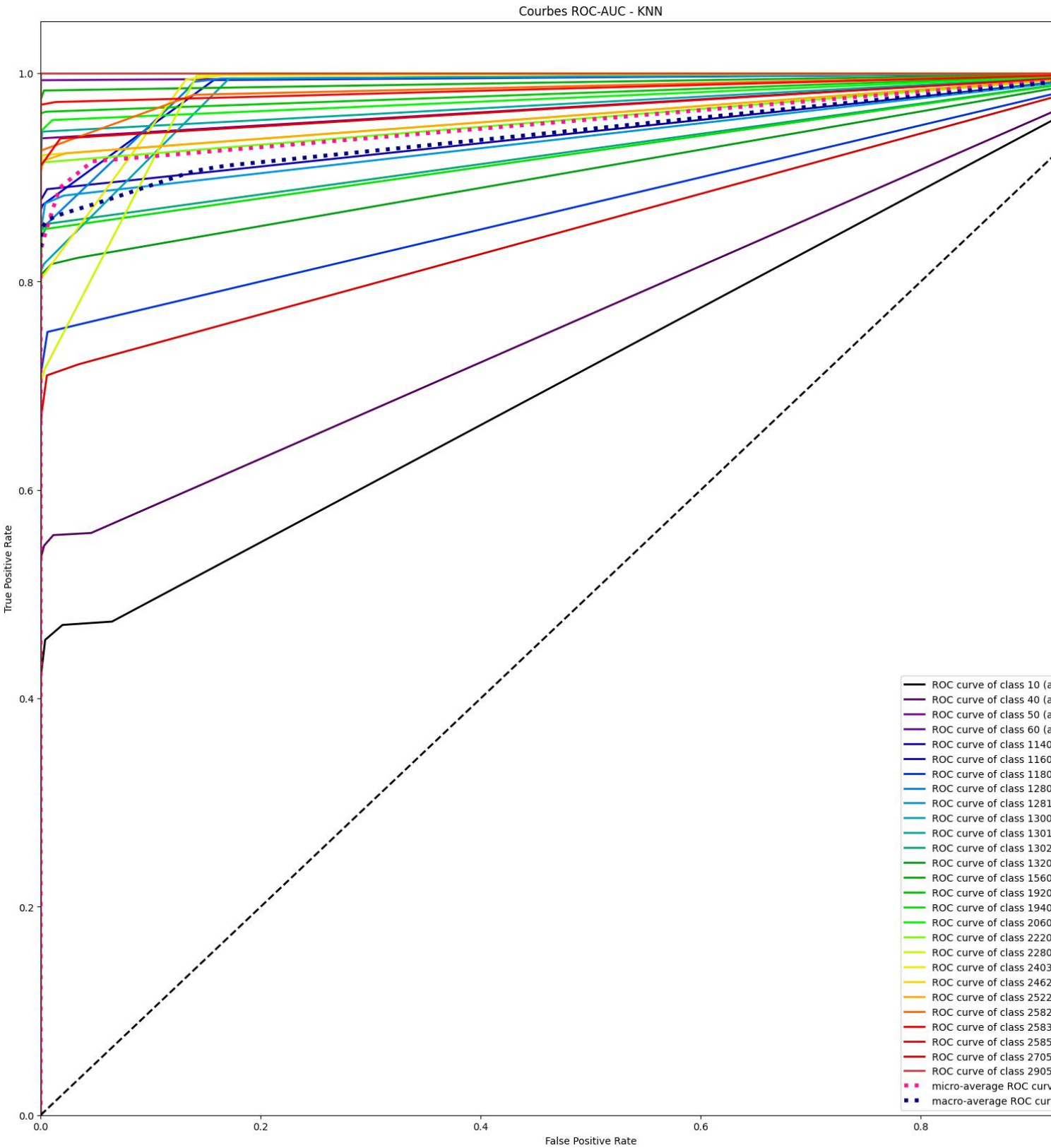
0.82783883, 0.96602492, 0.89240506, 0.80282519, 0.88643881,

0.94339623, 0.94900698, 0.95010846, 0.98031915, 0.94355698,

0.79162304, 1.])]

train_mse_result = 160215.49750805323

test_mse_result = 184986.59539293745



KNN (150 WORDS BY CODE)

ESTIMATOR KNEIGHBORSCLASSIFIER()

PARAMS {'N_NEIGHBORS': [10]}

TRAIN_R2_SCORE = 0.8857199294961405

TEST_R2_SCORE = 0.8786847383455905

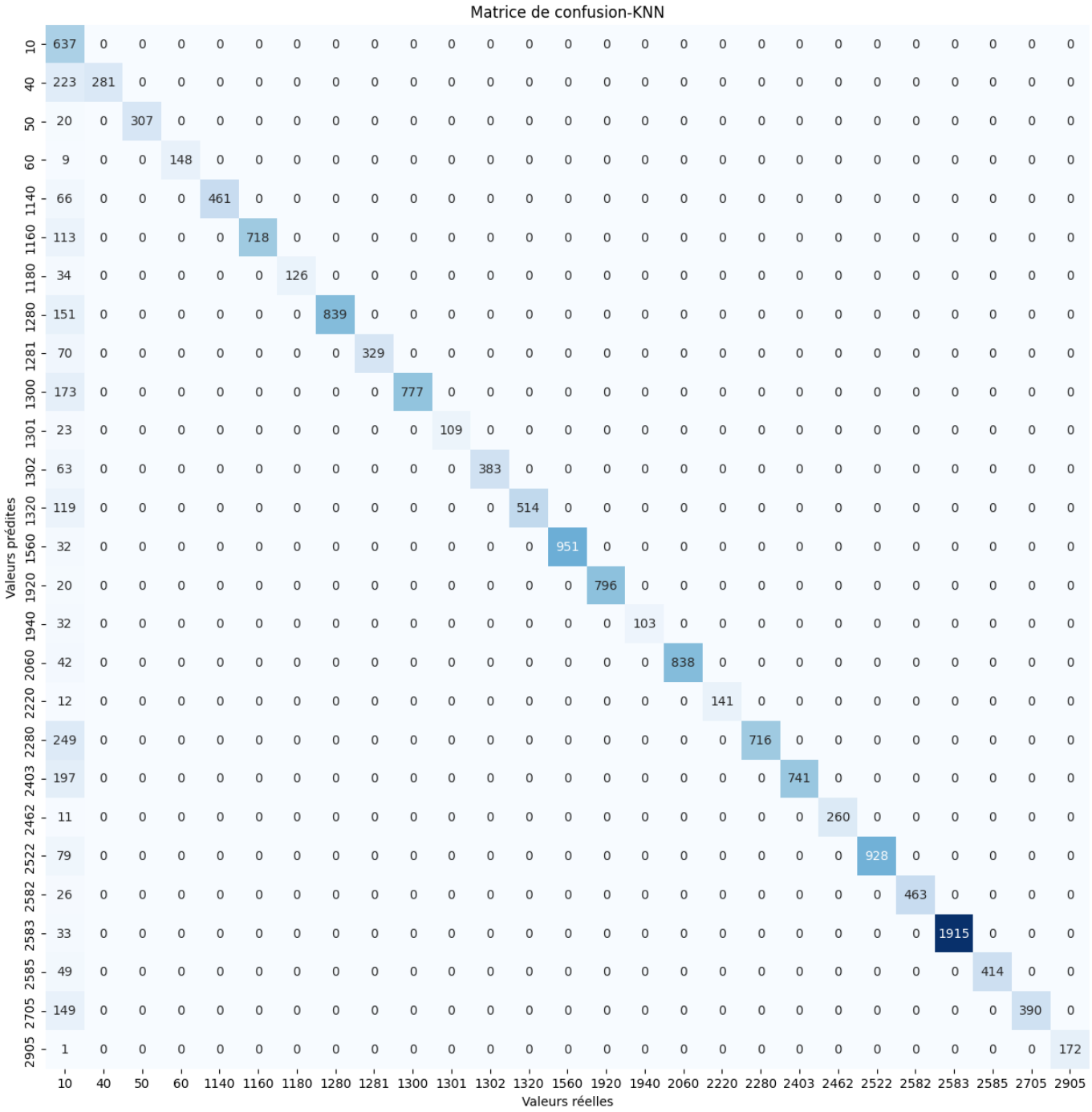
X_train.shape - X_test.shape - len(y_train) - len(y_test)

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

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

==========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,

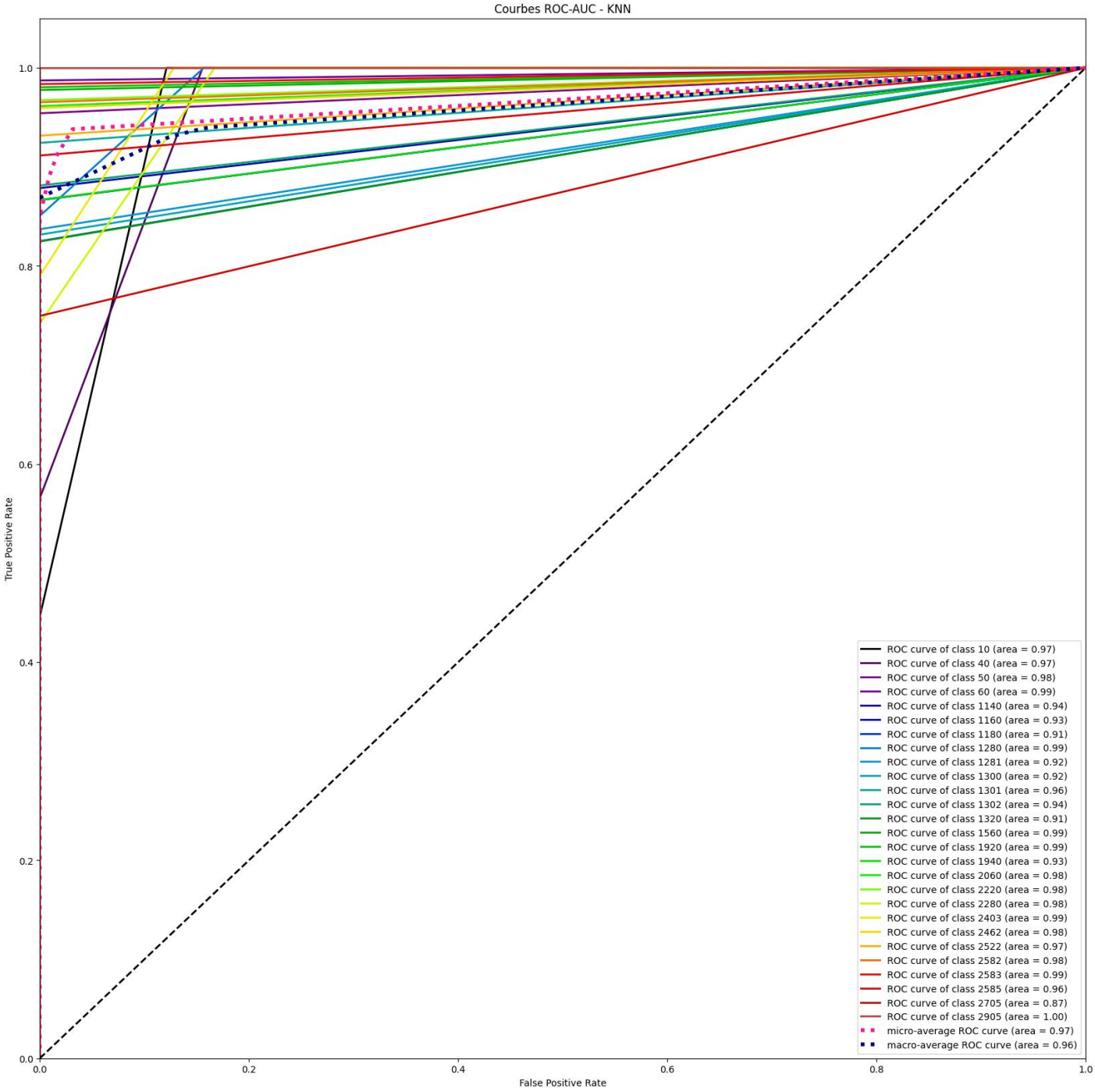
0.97928437, 0.95917313, 0.97268908, 0.99145742, 0.94412771,

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]}

TRAIN_R2_SCORE = 0.8887436941591199

TEST_R2_SCORE = 0.88160213942746

---

df.shape : (82265, 4052)

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

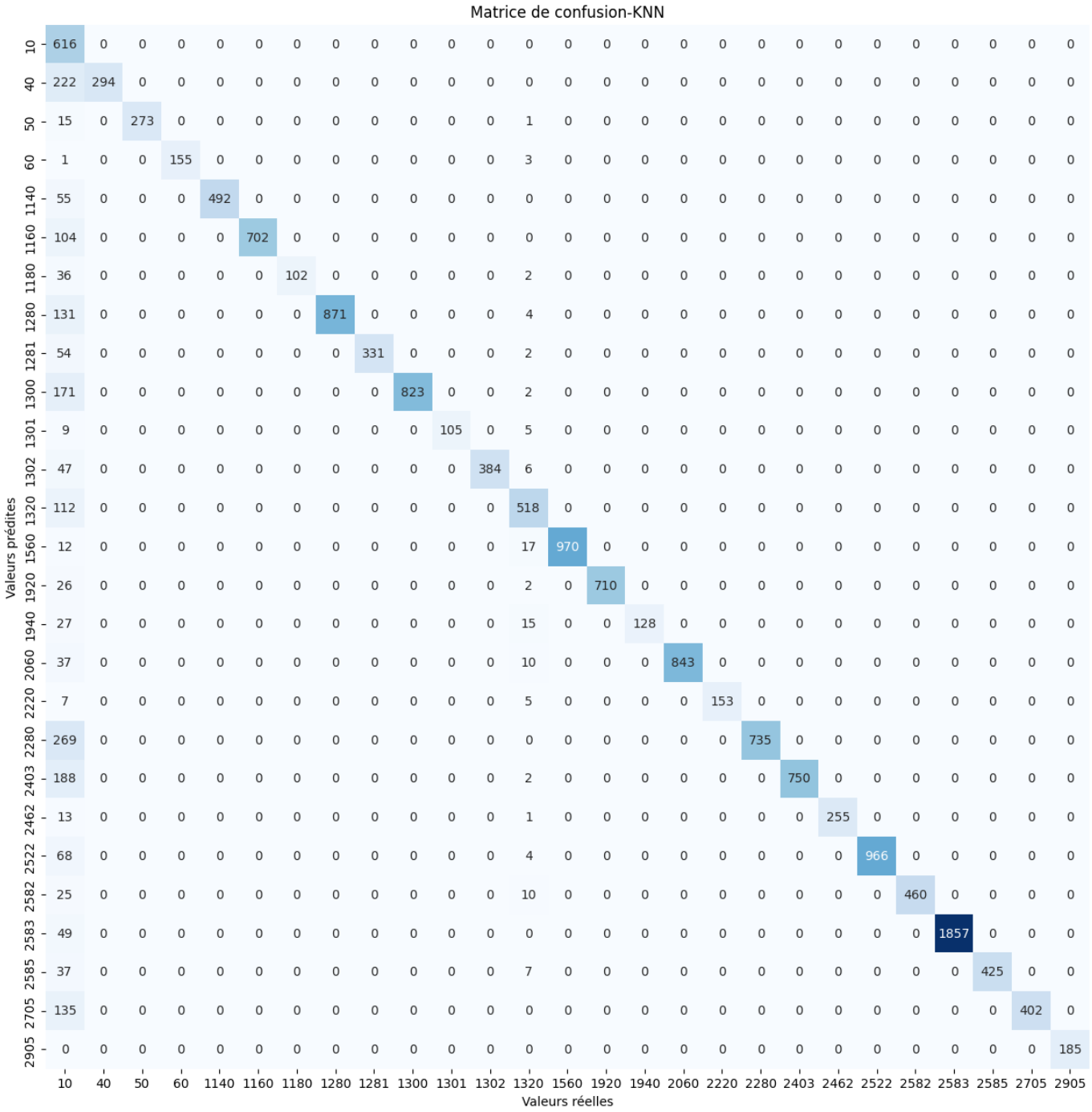
X_train.shape - X_test.shape - len(y_train) - len(y_test)

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

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

Use SEABORN to draw confusion_matrix----------

Confusion matrix as graph with Seaborn :



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)

TRAIN_R2_SCORE = 0.8861802979450039

TEST_R2_SCORE = 0.8843028732925106

BEST_PARAMS: [{'ALGORITHM': 'AUTO', 'N_JOBS': -1, 'N_NEIGHBORS': 10, 'WEIGHTS': 'DISTANCE'}]

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_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])]

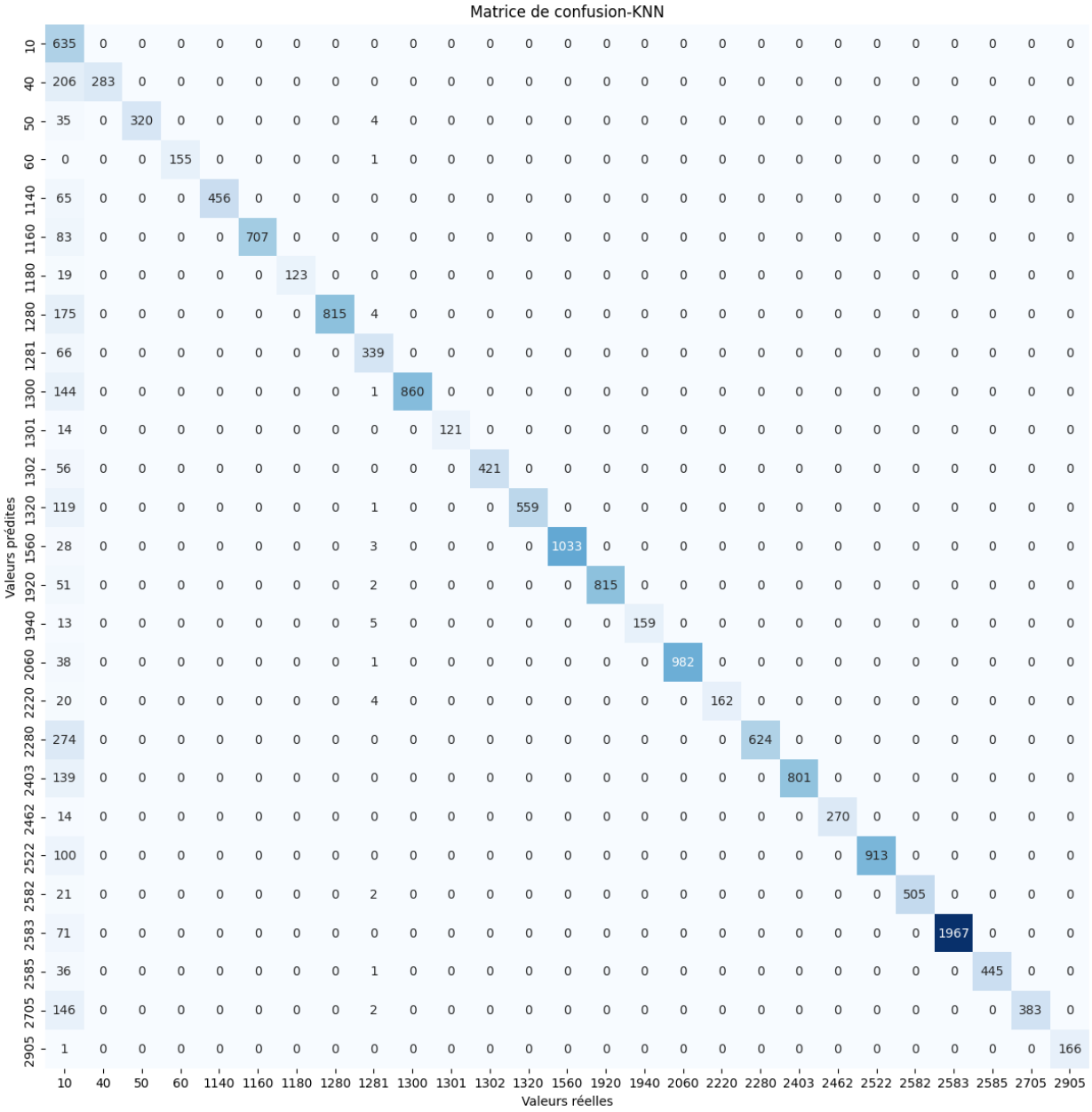
train_mse_result = 382045.8192162751

test_mse_result = 388084.74004945834

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

Use SEABORN to draw confusion_matrix----------

Confusion matrix as graph with Seaborn :



KNN (100 WORDS BY CODE) APRES UNE PCA (REDUCTION DE 80% DES VARIABLES) – 2MIN

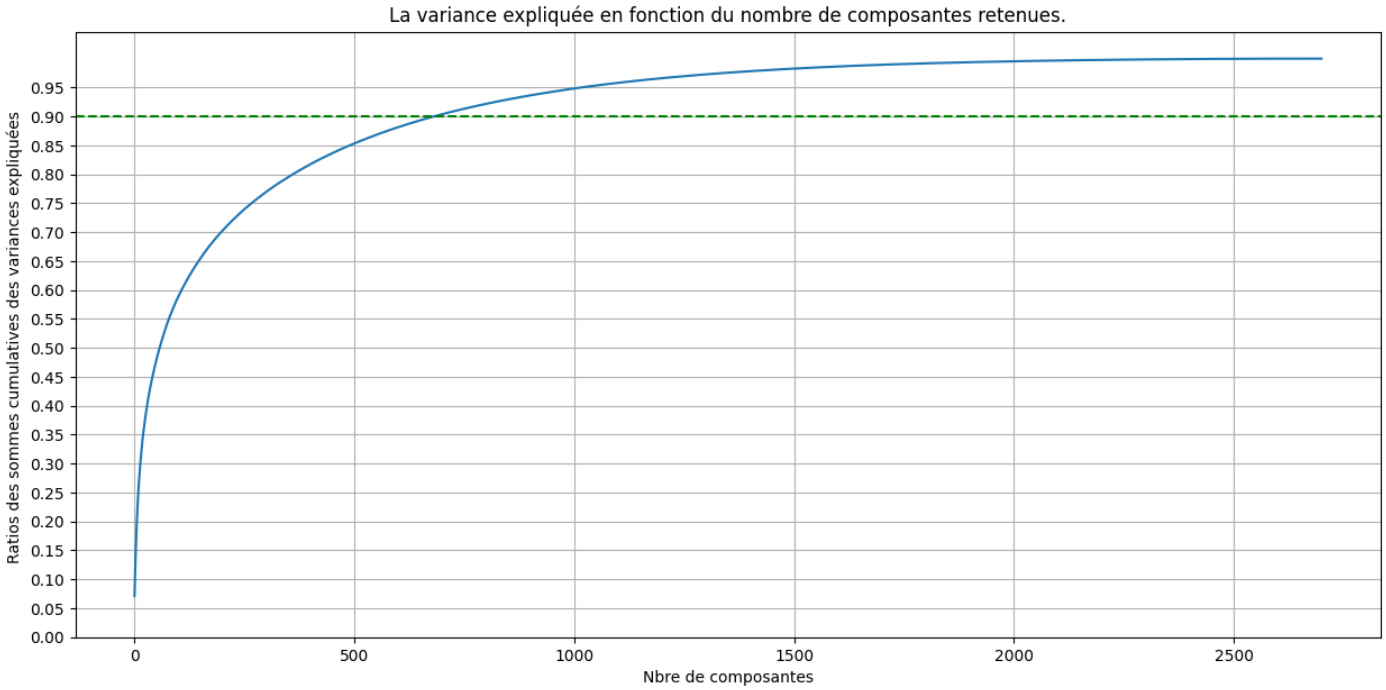
TRAIN_R2_SCORE = 0.8463046252962986

TEST_R2_SCORE = 0.8340120342794627

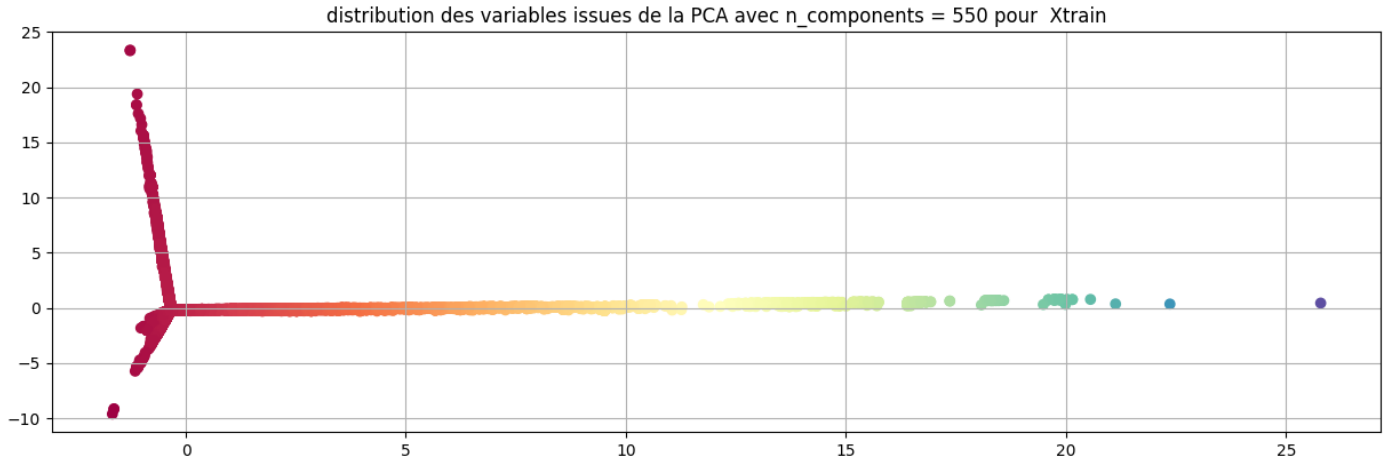
TRAIN_MSE_RESULT = 409521.15333069954

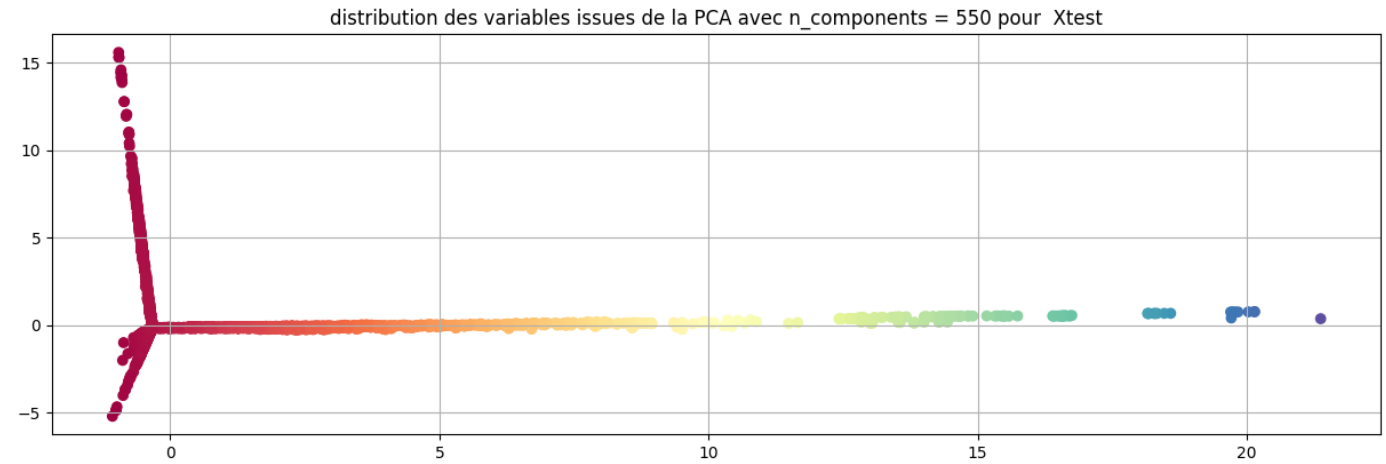
TEST_MSE_RESULT = 195946.71190664318

BEST_PARAMS: [{'N_NEIGHBORS': 10}]



Un minimum de **550** pour le # de composantes après réduction de dimensions PCA donnant un pourcentage de réduction de : **80.0** %

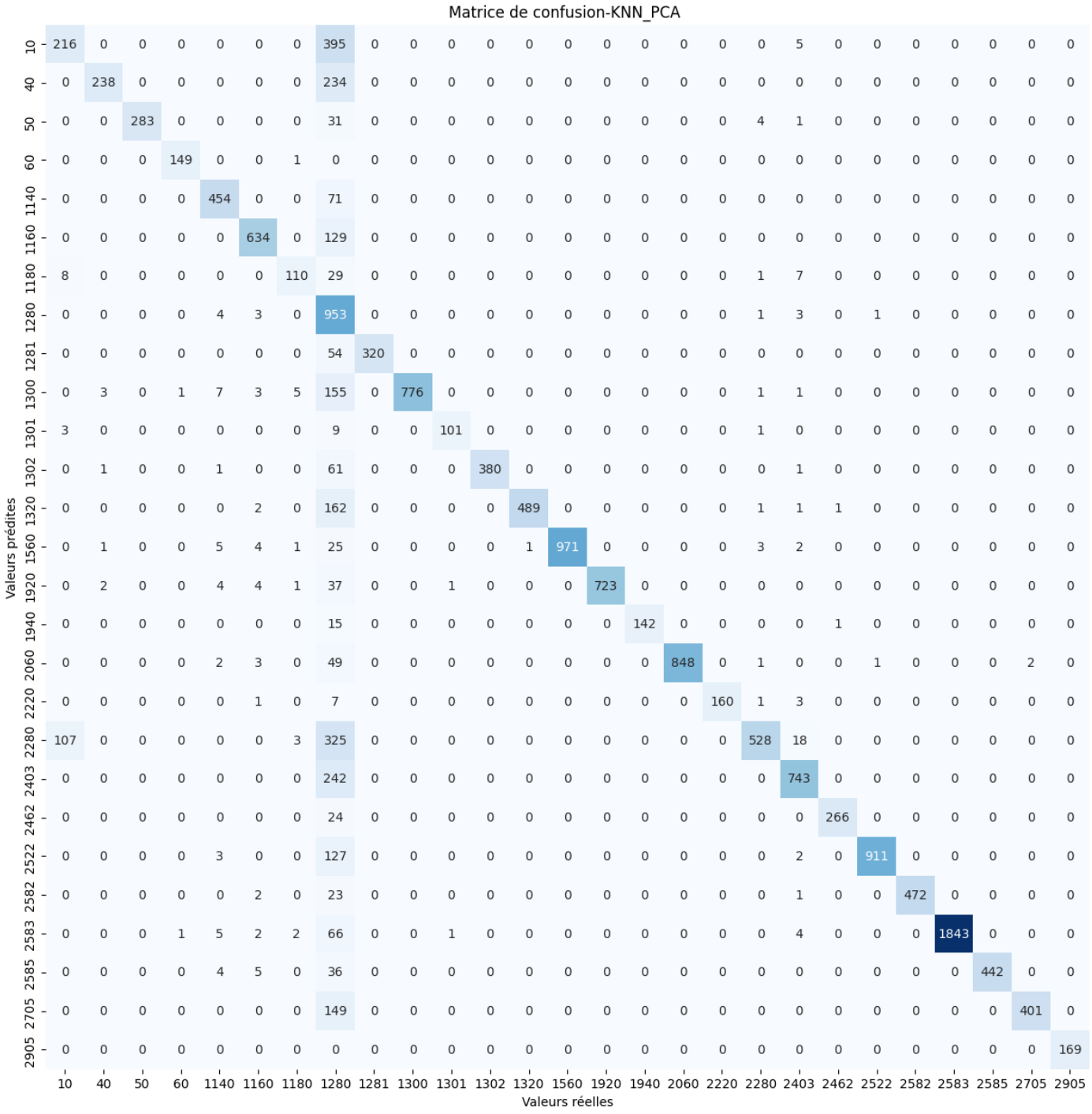


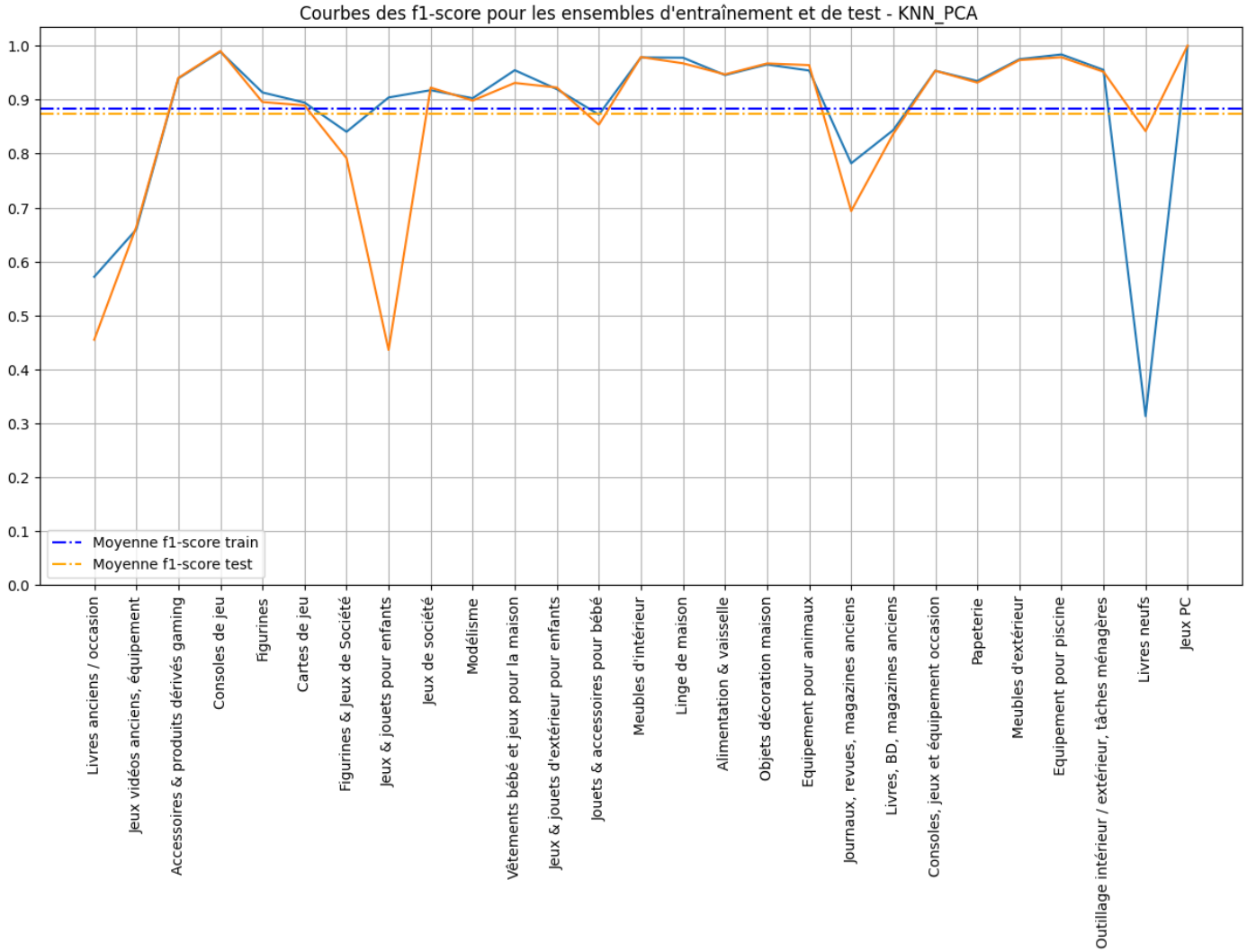


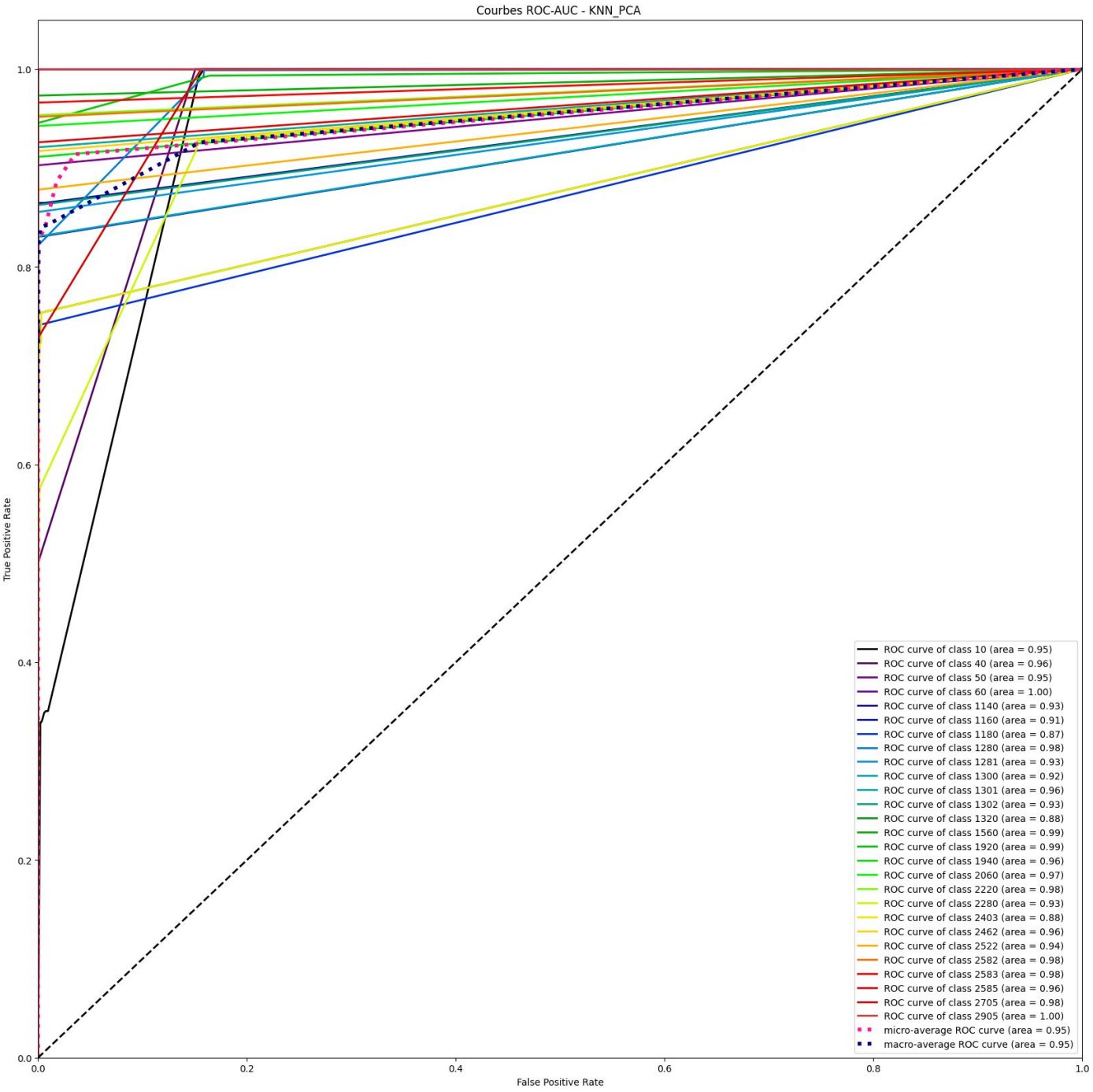
df.shape : (82265, 2702)

X_train.shape - X_test.shape - len(y_train) - len(y_test)
(65812, 550) - (16453, 550) - 65812 – 16453

{'mean_fit_time': array([0.18550777]),
'std_fit_time': array([0.00195567]),
'mean_score_time': array([30.12307461]),
'std_score_time': array([0.06649759]),
'param_n_neighbors': masked_array(data=[10],
mask=[False],
fill_value='?',
dtype=object),
'params': [{'n_neighbors': 10}],
'split0_test_score': array([0.84086972]),
'split1_test_score': array([0.84159183]),
'split2_test_score': array([0.83753476]),
'mean_test_score': array([0.83999877]),
'std_test_score': array([0.00176708]),
'rank_test_score': array([1])}




Courbes des f1-score pour les ensembles d'entraînement et de test - KNN_PCA
Moyenne f1-score train
Moyenne f1-score test
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Livres anciens / occasion
Jeux vidéos anciens, équipement
Accessoires & produits dérivés gaming
Consoles de jeu
Figurines
Cartes de jeu
Figurines & Jeux de Société
Jeux & jouets pour enfants
Jeux de société
Modélisme
Vêtements bébé et jeux pour la maison
Jeux & jouets d'extérieur pour enfants
Jouets & accessoires pour bébé
Meubles d'intérieur
Linge de maison
Alimentation & vaisselle
Objets décoration maison
Equipement pour animaux
Journaux, revues, magazines anciens
Livres, BD, magazines anciens
Consoles, jeux et équipement occasion
Papeterie
Meubles d'extérieur
Equipement pour piscine
Outillage intérieur / extérieur, tâches ménagères
Livres neufs
Jeux PC


Courbes ROC-AUC - KNN_PCA
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 10 (area = 0.95)
ROC curve of class 40 (area = 0.96)
ROC curve of class 50 (area = 0.95)
ROC curve of class 60 (area = 1.00)
ROC curve of class 1140 (area = 0.93)
ROC curve of class 1160 (area = 0.91)
ROC curve of class 1180 (area = 0.87)
ROC curve of class 1280 (area = 0.98)
ROC curve of class 1281 (area = 0.93)
ROC curve of class 1300 (area = 0.92)
ROC curve of class 1301 (area = 0.96)
ROC curve of class 1302 (area = 0.93)
ROC curve of class 1320 (area = 0.88)
ROC curve of class 1560 (area = 0.99)
ROC curve of class 1920 (area = 0.99)
ROC curve of class 1940 (area = 0.96)
ROC curve of class 2060 (area = 0.97)
ROC curve of class 2220 (area = 0.98)
ROC curve of class 2280 (area = 0.93)
ROC curve of class 2403 (area = 0.88)
ROC curve of class 2462 (area = 0.96)
ROC curve of class 2522 (area = 0.94)
ROC curve of class 2582 (area = 0.98)
ROC curve of class 2583 (area = 0.98)
ROC curve of class 2585 (area = 0.96)
ROC curve of class 2705 (area = 0.98)
ROC curve of class 2905 (area = 1.00)
micro-average ROC curve (area = 0.95)
macro-average ROC curve (area = 0.95)

RFC - RANDOMFORESTCLASSIFIER (300 WORDS BY CODE) – THE BEST

TRAIN_R2_SCORE = 0.9220203002491947

TEST_R2_SCORE = 0.9121740715978849

BEST_PARAMS: [{'MAX_FEATURES': 'SQRT', 'MIN_SAMPLES_SPLIT': 10}]

X_train.shape - X_test.shape - len(y_train) - len(y_test)

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

estimator RandomForestClassifier()

params {'max_features': ['sqrt'], 'min_samples_split'...

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

train_f1_score = [array([0.49328594, 0.79447115, 0.98415153, 0.99516908, 0.96825397,

0.95305318, 0.92307692, 0.94928335, 0.95154472, 0.92915893,

0.9894958 , 0.95852018, 0.93389297, 0.99550302, 0.99721813,

0.98020586, 0.98858892, 0.98505114, 0.89335485, 0.9163918 ,

0.98128708, 0.97629708, 0.99320071, 0.99503514, 0.97842105,

0.87859506, 1.])]

test_f1_score = [array([0.46021666, 0.76601307, 0.97592295, 1. , 0.95626243,

0.93954135, 0.9122807 , 0.94246575, 0.94200849, 0.9255079 ,

0.97297297, 0.94911243, 0.92193919, 0.99454094, 0.99413681,

0.95709571, 0.98487395, 0.97313433, 0.90145577, 0.91160221,

0.95683453, 0.97393015, 0.98263534, 0.99424987, 0.97473684,

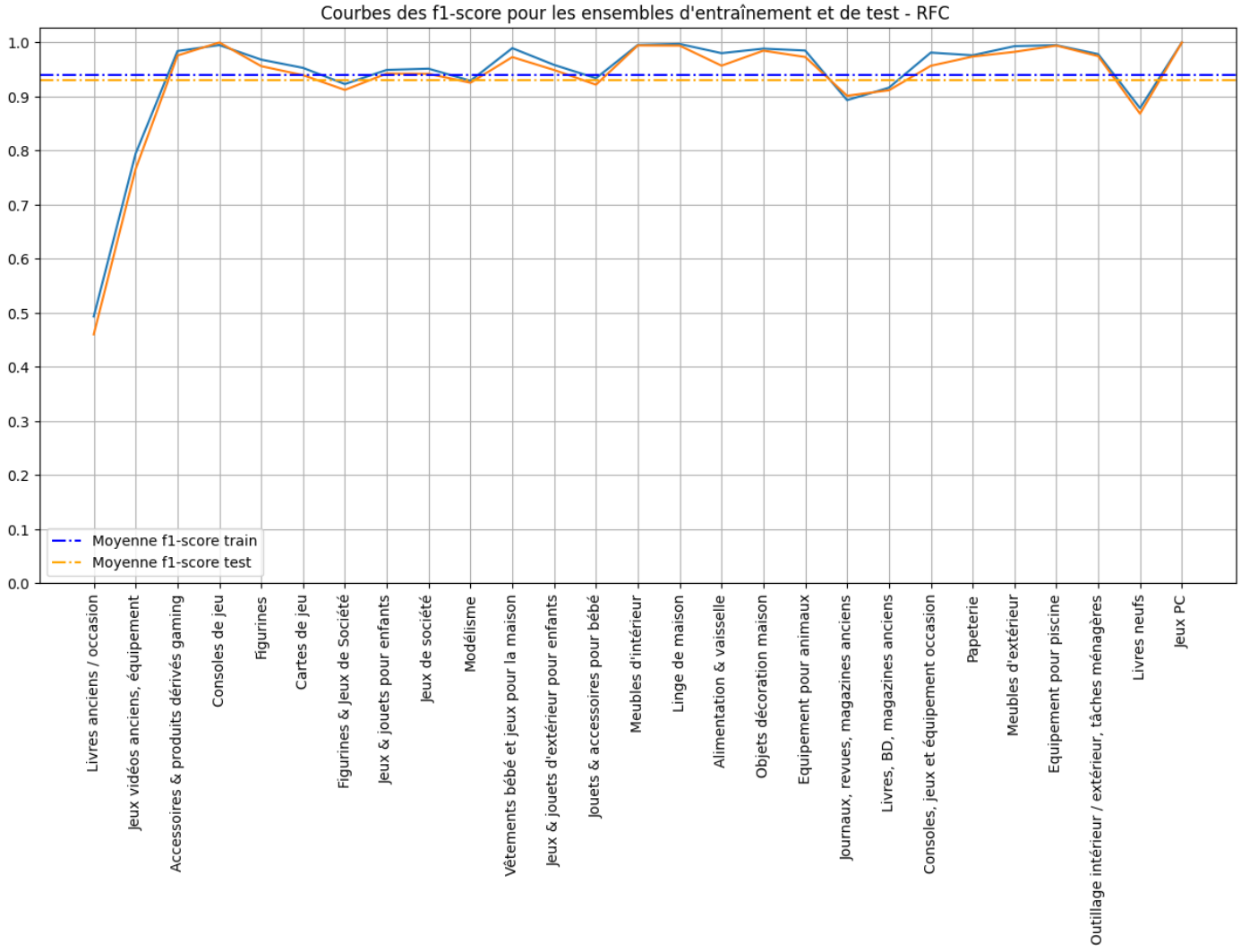
0.86831276, 1.])]

mean_train_f1_score= 0.9400928726242548

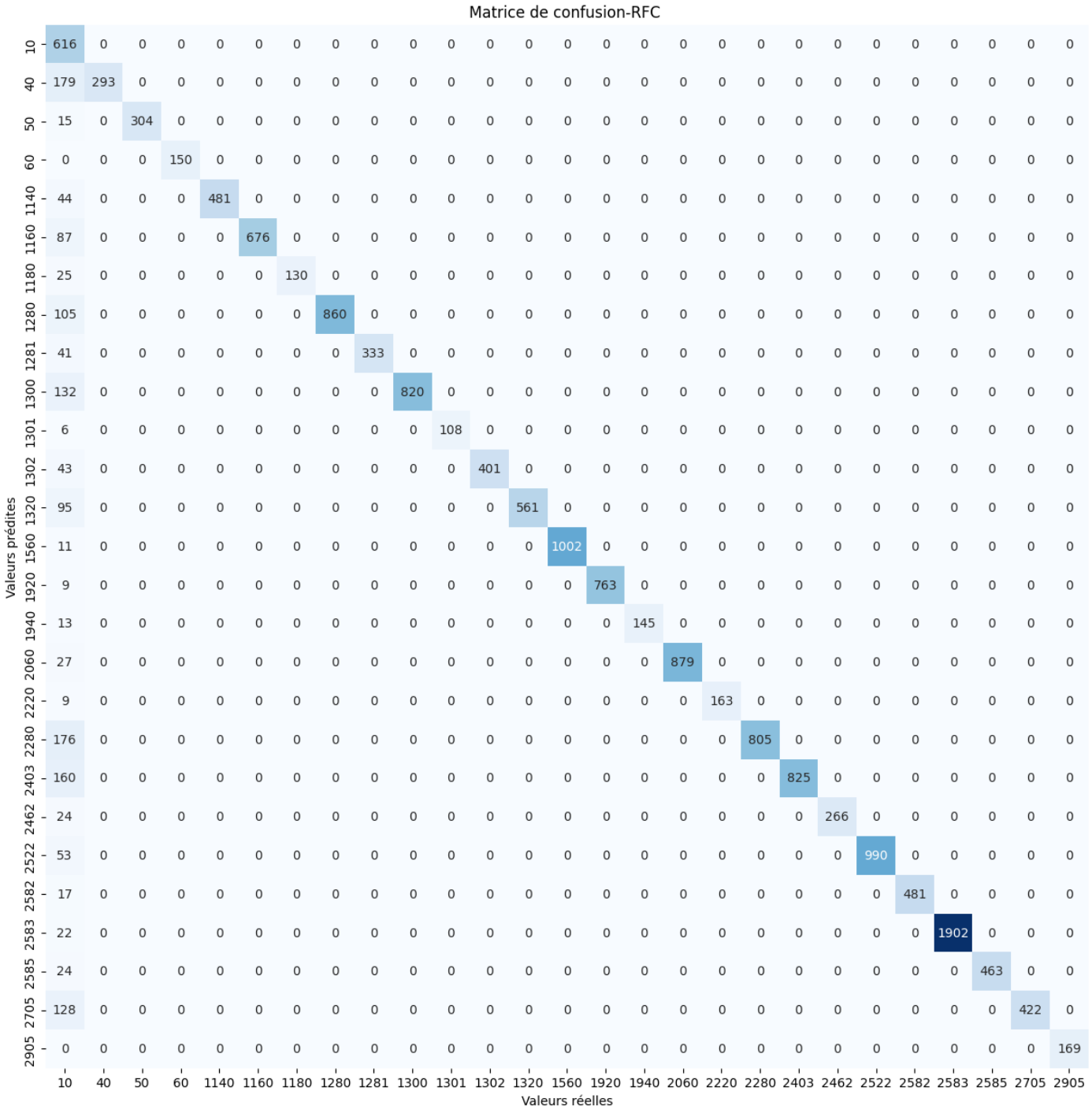
mean_test_f1_score= 0.9308067817223938

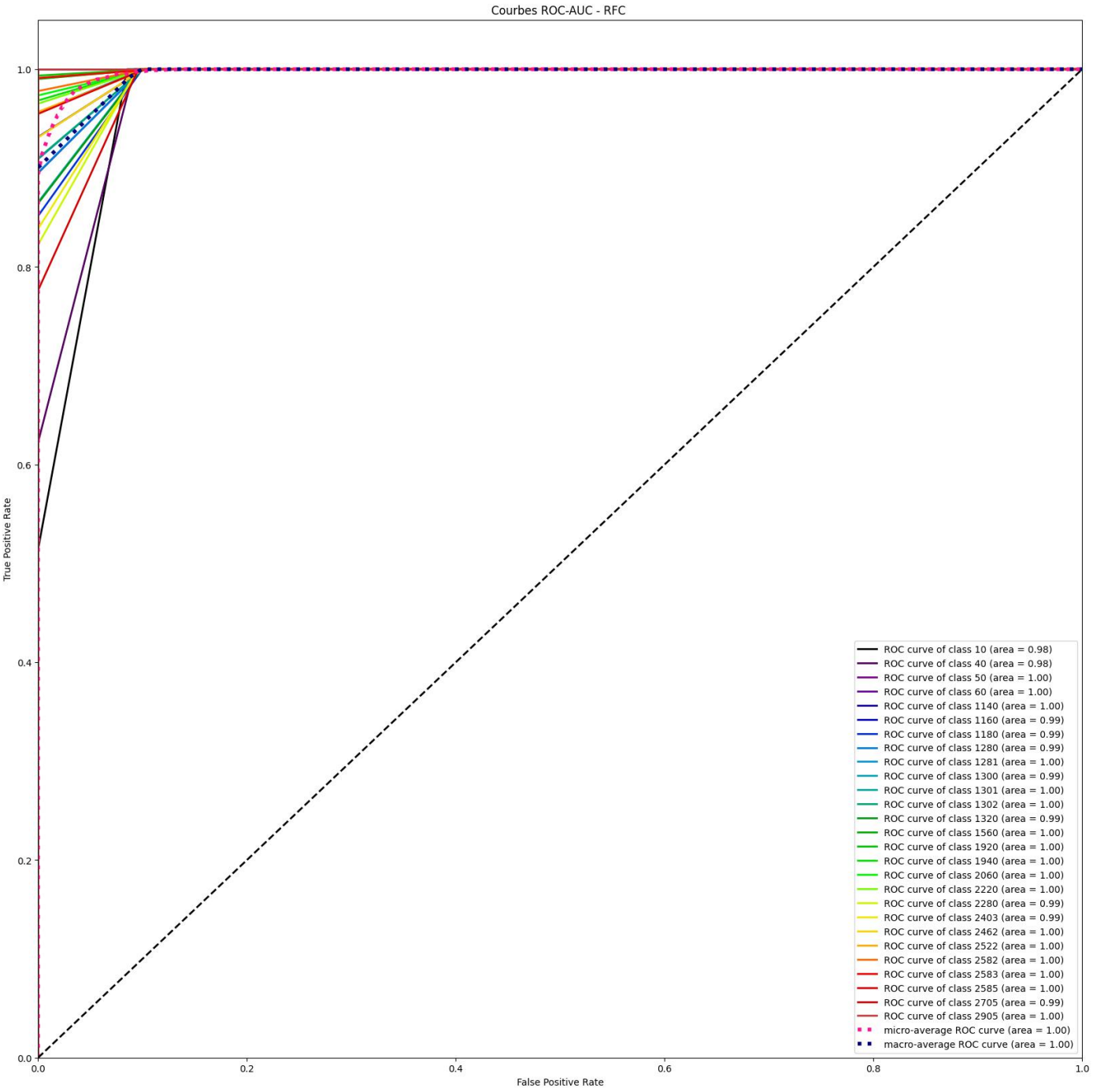
	precision	recall	f1-score	support
10	0.30	1.00	0.46	616
40	1.00	0.62	0.77	472
50	1.00	0.95	0.98	319
60	1.00	1.00	1.00	150
1140	1.00	0.92	0.96	525
1160	1.00	0.89	0.94	763
1180	1.00	0.84	0.91	155
1280	1.00	0.89	0.94	965
1281	1.00	0.89	0.94	374

1300	1.00	0.86	0.93	952
1301	1.00	0.95	0.97	114
1302	1.00	0.90	0.95	444
1320	1.00	0.86	0.92	656
1560	1.00	0.99	0.99	1013
1920	1.00	0.99	0.99	772
1940	1.00	0.92	0.96	158
2060	1.00	0.97	0.98	906
2220	1.00	0.95	0.97	172
2280	1.00	0.82	0.90	981
2403	1.00	0.84	0.91	985
2462	1.00	0.92	0.96	290
2522	1.00	0.95	0.97	1043
2582	1.00	0.97	0.98	498



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


Matrice de confusion-RFC
Valeurs prédites
Valeurs réelles
10 40 50 60 1140 1160 1180 1280 1281 1300 1301 1302 1320 1560 1920 1940 2060 2220 2280 2403 2462 2522 2582 2583 2585 2705 2905


Courbes ROC-AUC - RFC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 10 (area = 0.98)
ROC curve of class 40 (area = 0.98)
ROC curve of class 50 (area = 1.00)
ROC curve of class 60 (area = 1.00)
ROC curve of class 1140 (area = 1.00)
ROC curve of class 1160 (area = 0.99)
ROC curve of class 1180 (area = 0.99)
ROC curve of class 1280 (area = 0.99)
ROC curve of class 1281 (area = 1.00)
ROC curve of class 1300 (area = 0.99)
ROC curve of class 1301 (area = 1.00)
ROC curve of class 1302 (area = 1.00)
ROC curve of class 1320 (area = 0.99)
ROC curve of class 1560 (area = 1.00)
ROC curve of class 1920 (area = 1.00)
ROC curve of class 1940 (area = 1.00)
ROC curve of class 2060 (area = 1.00)
ROC curve of class 2220 (area = 1.00)
ROC curve of class 2280 (area = 0.99)
ROC curve of class 2403 (area = 0.99)
ROC curve of class 2462 (area = 1.00)
ROC curve of class 2522 (area = 1.00)
ROC curve of class 2582 (area = 1.00)
ROC curve of class 2583 (area = 1.00)
ROC curve of class 2585 (area = 1.00)
ROC curve of class 2705 (area = 0.99)
ROC curve of class 2905 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)

LREG (100 WORDS BY CODE) – 4MIN

BEST_PARAMS: [{'C': 30}]

TRAIN_R2_SCORE = 0.8658603294232055

TEST_R2_SCORE = 0.8622135780708685

X_train.shape - X_test.shape - len(y_train) - len(y_test)

(65812, 2700) - (16453, 2700) - 65812 - 16453

estimator LogisticRegression()

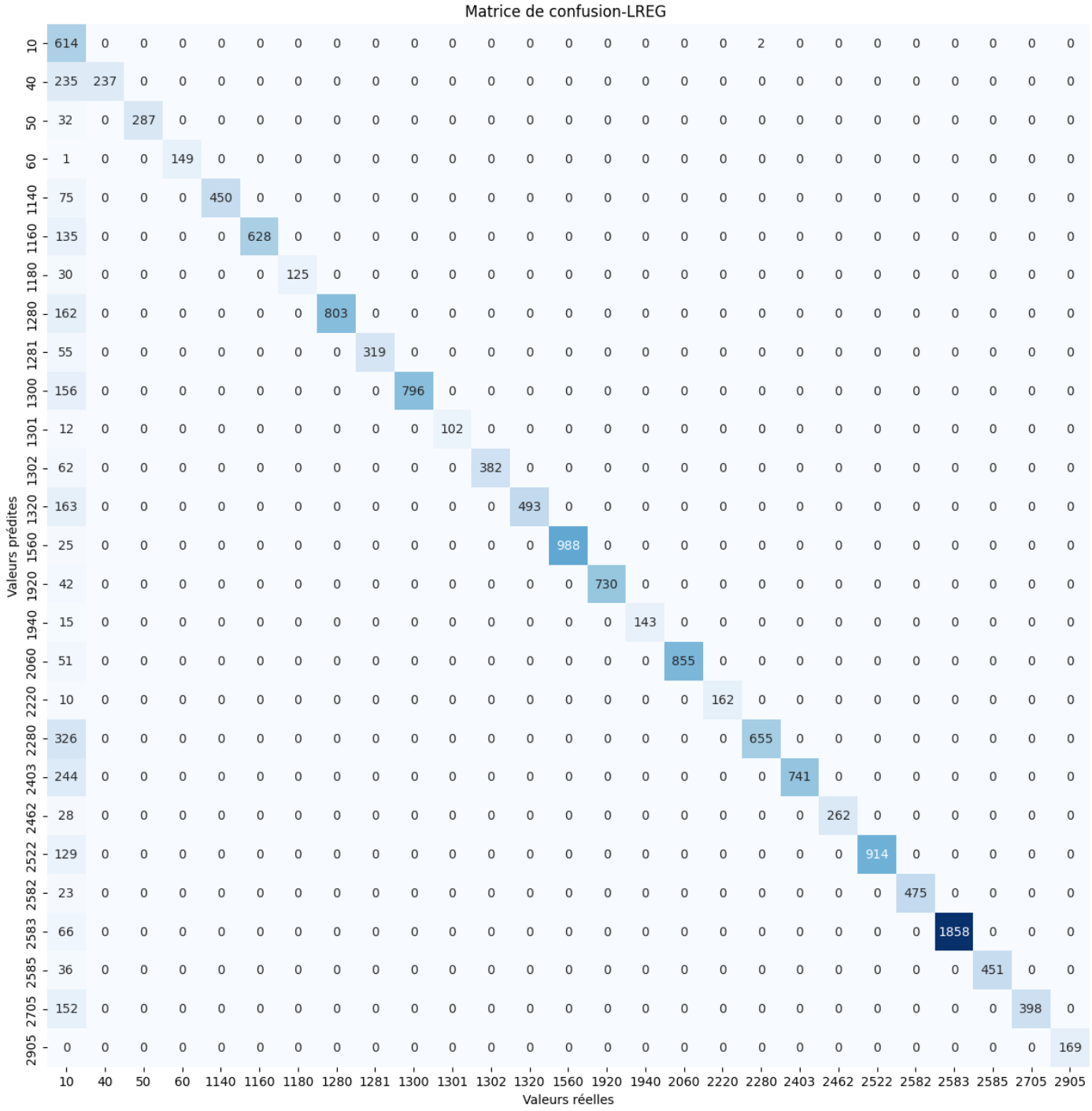
params {'C': [5, 10, 20]}

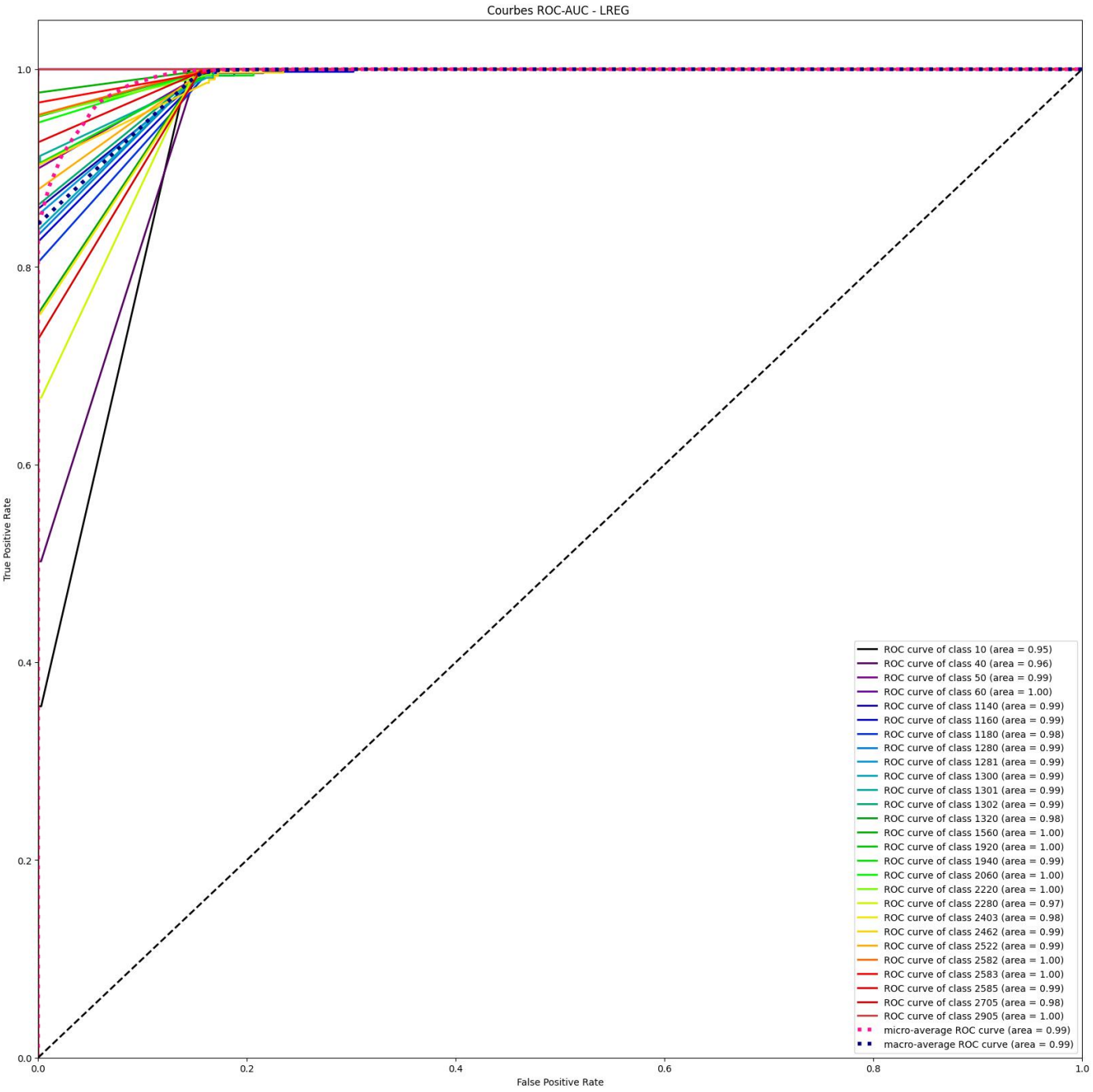
train_f1_score = [array([0.36140046, 0.66355763, 0.94627105, 0.99273608, 0.93363162,
0.9073154 , 0.89071038, 0.9119452 , 0.91848373, 0.90918919,
0.9622438 , 0.92756133, 0.87660327, 0.98651802, 0.98189068,
0.95114007, 0.97431555, 0.96634615, 0.79063803, 0.85167173,
0.96040987, 0.93652531, 0.98114169, 0.98680361, 0.96119882,
0.83593131, 1.])]

test_f1_score = [array([0.35135908, 0.66854725, 0.94719472, 0.99665552, 0.92307692,
0.90294752, 0.89285714, 0.90837104, 0.92063492, 0.91075515,
0.94444444, 0.92493947, 0.85813751, 0.98750625, 0.97203728,
0.95016611, 0.97103918, 0.97005988, 0.7997558 , 0.85863268,
0.94927536, 0.93408278, 0.97636177, 0.98254892, 0.96162047,
0.83966245, 1.])]

train_mse_result = 456855.5308302437

test_mse_result = 477873.36564760224




Courbes ROC-AUC - LREG
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 10 (area = 0.95)
ROC curve of class 40 (area = 0.96)
ROC curve of class 50 (area = 0.99)
ROC curve of class 60 (area = 1.00)
ROC curve of class 1140 (area = 0.99)
ROC curve of class 1160 (area = 0.99)
ROC curve of class 1180 (area = 0.98)
ROC curve of class 1280 (area = 0.99)
ROC curve of class 1281 (area = 0.99)
ROC curve of class 1300 (area = 0.99)
ROC curve of class 1301 (area = 0.99)
ROC curve of class 1302 (area = 0.99)
ROC curve of class 1320 (area = 0.98)
ROC curve of class 1560 (area = 1.00)
ROC curve of class 1920 (area = 1.00)
ROC curve of class 1940 (area = 0.99)
ROC curve of class 2060 (area = 1.00)
ROC curve of class 2220 (area = 1.00)
ROC curve of class 2280 (area = 0.97)
ROC curve of class 2403 (area = 0.98)
ROC curve of class 2462 (area = 0.99)
ROC curve of class 2522 (area = 0.99)
ROC curve of class 2582 (area = 1.00)
ROC curve of class 2583 (area = 1.00)
ROC curve of class 2585 (area = 0.99)
ROC curve of class 2705 (area = 0.98)
ROC curve of class 2905 (area = 1.00)
micro-average ROC curve (area = 0.99)
macro-average ROC curve (area = 0.99)

RBF (100 WORDS BY CODE)

TRAIN_R2_SCORE = 0.8660274721935209

TEST_R2_SCORE = 0.8619704613140461

BEST_PARAMS: [{'MAX_FEATURES': 'SQRT', 'MIN_SAMPLES_SPLIT': 10}]

X_train.shape - X_test.shape - len(y_train) - len(y_test)

(65812, 2700) - (16453, 2700) - 65812 - 16453

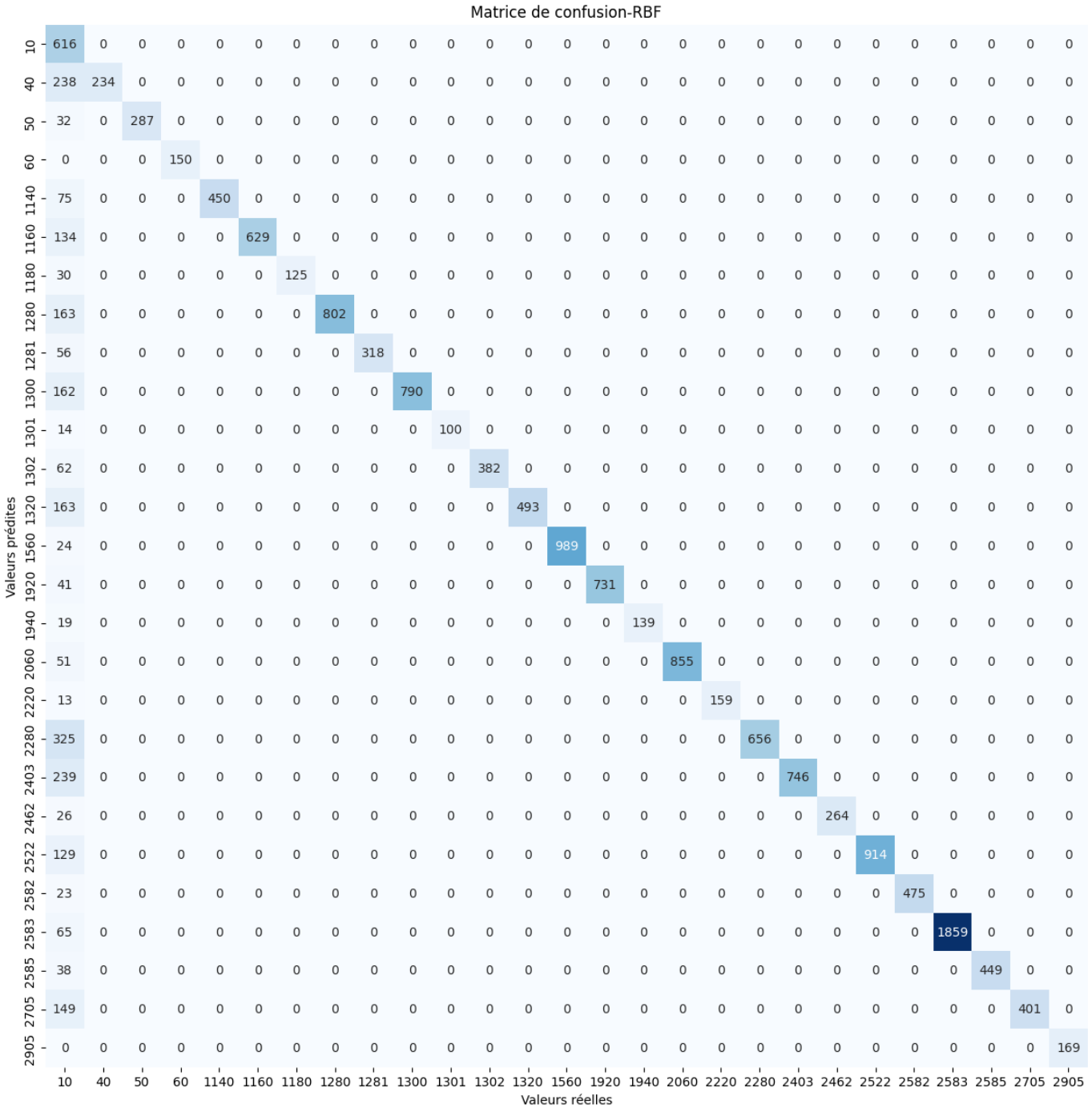
estimator RandomForestClassifier()

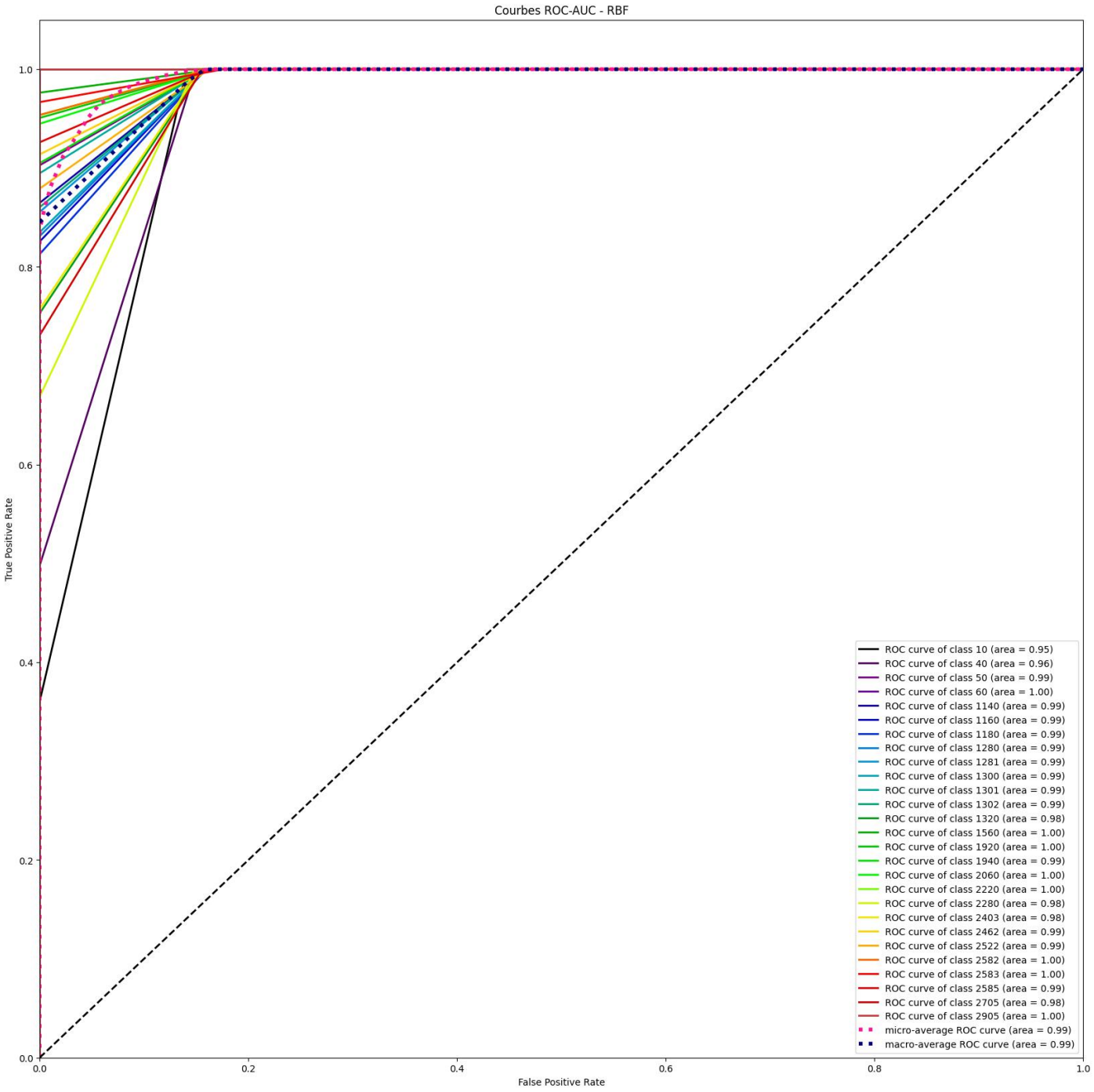
```
params    {'name': 'RBF', 'estimator': ensemble.RandomForestClassifier(), 'params': {'max_features': ["sqrt", None],
                'min_samples_split': [1, 10]}
            },
           {'name': 'SVC',  'estimator': svm.SVC(),
            'params': {'kernel':('linear', 'rbf'), 'C':[1, 10]}
            }
```

```
train_f1_score = [array([0.36168826, 0.66088117, 0.94627105, 0.99273608, 0.93363162,
       0.9073154 , 0.89071038, 0.9119452 , 0.91848373, 0.90918919,
       0.9622438 , 0.92756133, 0.87660327, 0.98651802, 0.98189068,
       0.94857143, 0.97431555, 0.96634615, 0.79063803, 0.85341426,
       0.96040987, 0.93725222, 0.98140127, 0.98680361, 0.96203209,
       0.83623877, 1.        ])]
test_f1_score = [array([0.35169854, 0.66288952, 0.94719472, 1.        , 0.92307692,
       0.90373563, 0.89285714, 0.90775325, 0.91907514, 0.90700344,
       0.93457944, 0.92493947, 0.85813751, 0.98801199, 0.97272122,
       0.93602694, 0.97103918, 0.96072508, 0.8014661 , 0.86192952,
       0.95306859, 0.93408278, 0.97636177, 0.98281787, 0.95940171,
       0.84332282, 1.        ])]
```

train_mse_result = 455162.75148909015

test_mse_result = 475895.7078344375


Matrice de confusion-RBF
Valeurs prédites
Valeurs réelles
10 40 50 60 1140 1160 1180 1280 1281 1300 1301 1302 1320 1560 1920 1940 2060 2220 2280 2403 2462 2522 2582 2583 2585 2705 2905
10 616 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
40 238 234 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
50 32 0 287 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
60 0 0 0 150 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1140 75 0 0 0 450 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1160 134 0 0 0 0 629 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1180 30 0 0 0 0 0 125 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1280 163 0 0 0 0 0 0 802 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1281 56 0 0 0 0 0 0 0 318 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1300 162 0 0 0 0 0 0 0 0 790 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1301 14 0 0 0 0 0 0 0 0 0 100 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1302 62 0 0 0 0 0 0 0 0 0 0 382 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1320 163 0 0 0 0 0 0 0 0 0 0 0 493 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1560 24 0 0 0 0 0 0 0 0 0 0 0 0 989 0 0 0 0 0 0 0 0 0 0 0 0 0
1920 41 0 0 0 0 0 0 0 0 0 0 0 0 0 731 0 0 0 0 0 0 0 0 0 0 0 0
1940 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 139 0 0 0 0 0 0 0 0 0 0 0
2060 51 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 855 0 0 0 0 0 0 0 0 0 0
2220 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 159 0 0 0 0 0 0 0 0 0
2280 325 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 656 0 0 0 0 0 0 0 0
2403 239 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 746 0 0 0 0 0 0 0
2462 26 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 264 0 0 0 0 0 0
2522 129 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 914 0 0 0 0 0
2582 23 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 475 0 0 0 0
2583 65 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1859 0 0 0
2585 38 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 449 0 0
2705 149 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 401 0
2905 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 0 169


Courbes ROC-AUC - RBF
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 10 (area = 0.95)
ROC curve of class 40 (area = 0.96)
ROC curve of class 50 (area = 0.99)
ROC curve of class 60 (area = 1.00)
ROC curve of class 1140 (area = 0.99)
ROC curve of class 1160 (area = 0.99)
ROC curve of class 1180 (area = 0.99)
ROC curve of class 1280 (area = 0.99)
ROC curve of class 1281 (area = 0.99)
ROC curve of class 1300 (area = 0.99)
ROC curve of class 1301 (area = 0.99)
ROC curve of class 1302 (area = 0.99)
ROC curve of class 1320 (area = 0.98)
ROC curve of class 1560 (area = 1.00)
ROC curve of class 1920 (area = 1.00)
ROC curve of class 1940 (area = 0.99)
ROC curve of class 2060 (area = 1.00)
ROC curve of class 2220 (area = 1.00)
ROC curve of class 2280 (area = 0.98)
ROC curve of class 2403 (area = 0.98)
ROC curve of class 2462 (area = 0.99)
ROC curve of class 2522 (area = 0.99)
ROC curve of class 2582 (area = 1.00)
ROC curve of class 2583 (area = 1.00)
ROC curve of class 2585 (area = 0.99)
ROC curve of class 2705 (area = 0.98)
ROC curve of class 2905 (area = 1.00)
micro-average ROC curve (area = 0.99)
macro-average ROC curve (area = 0.99)

NAIVE BAYES (100 WORDS BY CODE) – 11SEC TEMPS D'EXCUCUTION

PARAMS {'ALPHA': [1]}

TRAIN_R2_SCORE = 0.8464261836747098

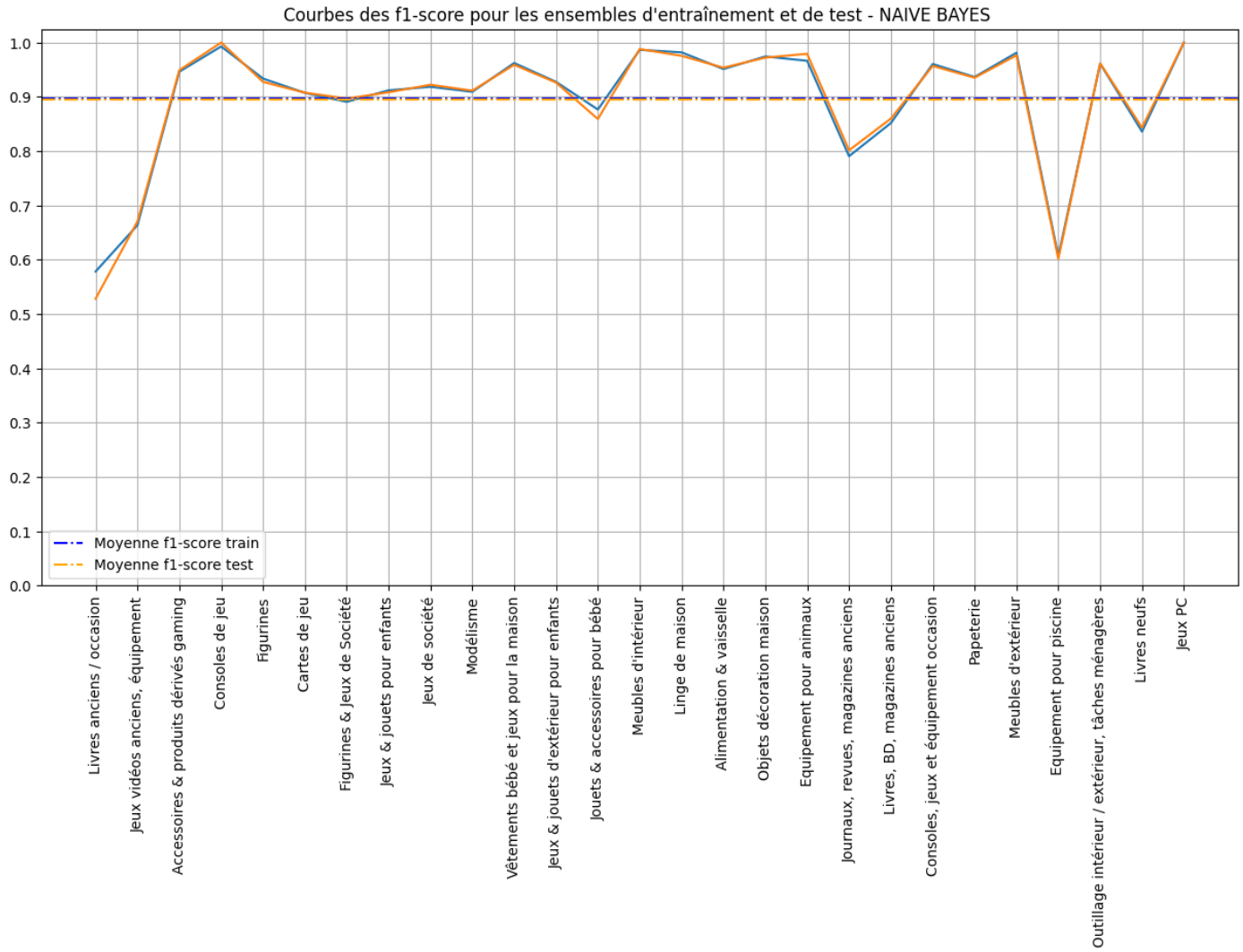
TEST_R2_SCORE = 0.8450738467148848

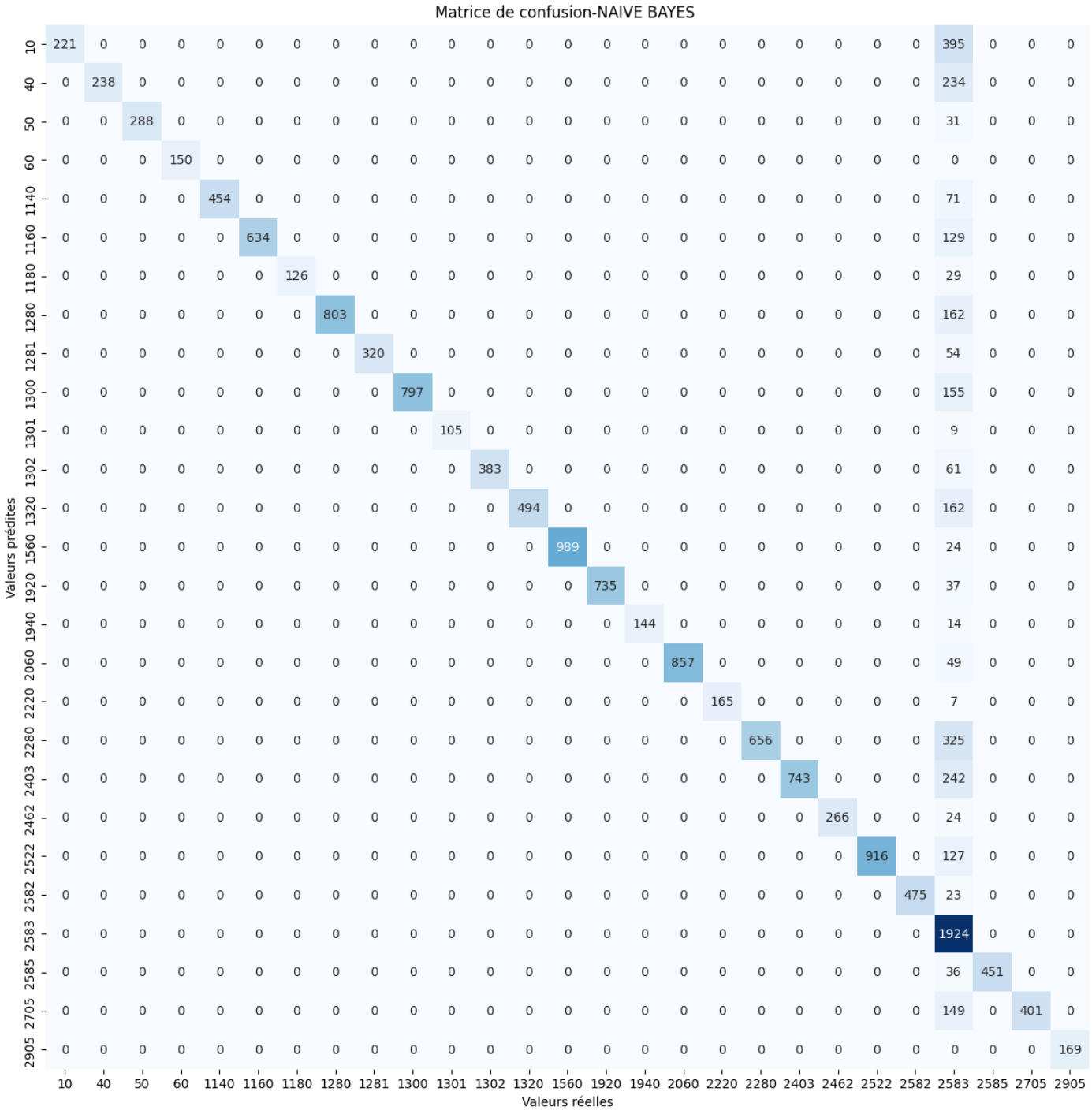
MEAN_TRAIN_F1_SCORE= 0.8964021730543796

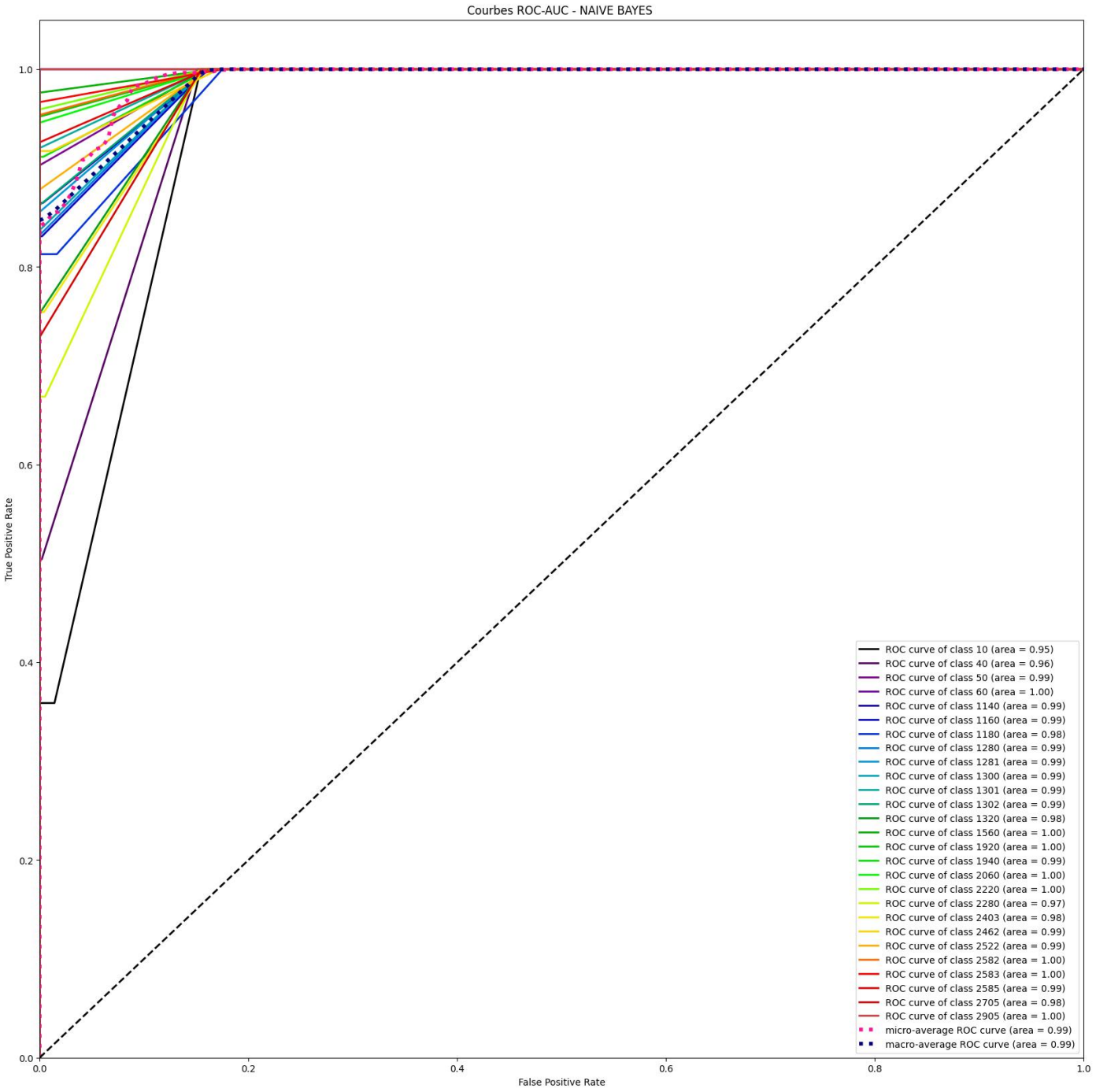
MEAN_TEST_F1_SCORE= 0.895199049313926

X_train.shape - X_test.shape - len(y_train) - len(y_test)

(65812, 2700) - (16453, 2700) - 65812 - 16453




Matrice de confusion-NAIVE BAYES
Valeurs prédites
Valeurs réelles
10 40 50 60 1140 1160 1180 1280 1281 1300 1301 1302 1320 1560 1920 1940 2060 2220 2280 2403 2462 2522 2582 2583 2585 2705 2905



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

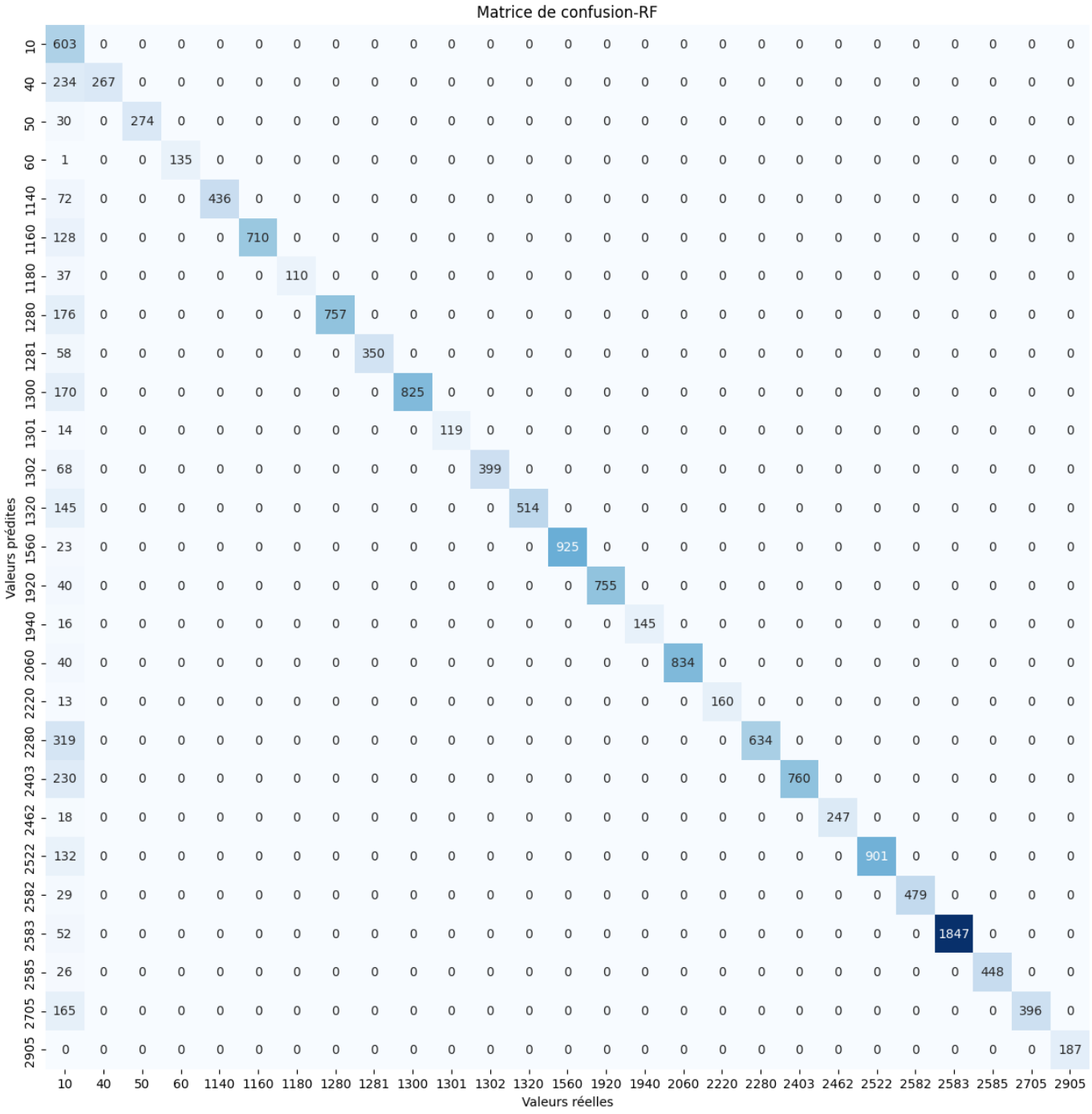
test_mse_result = 465541.4230231569

best_params: [{'max_features': 'sqrt', 'min_samples_split': 100}]

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

3. Use SEABORN to draw confusion_matrix----------

Confusion matrix as graph with Seaborn :


Matrice de confusion-RF
Valeurs prédites
Valeurs réelles
10 40 50 60 1140 1160 1180 1280 1281 1300 1301 1302 1320 1560 1920 1940 2060 2220 2280 2403 2462 2522 2582 2583 2585 2705 2905
603 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
234 267 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
30 0 274 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 135 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
72 0 0 0 436 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
128 0 0 0 0 710 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
37 0 0 0 0 0 110 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
176 0 0 0 0 0 0 757 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
58 0 0 0 0 0 0 0 350 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
170 0 0 0 0 0 0 0 0 825 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
14 0 0 0 0 0 0 0 0 0 119 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
68 0 0 0 0 0 0 0 0 0 0 399 0 0 0 0 0 0 0 0 0 0 0 0 0 0
145 0 0 0 0 0 0 0 0 0 0 0 514 0 0 0 0 0 0 0 0 0 0 0 0 0
23 0 0 0 0 0 0 0 0 0 0 0 0 925 0 0 0 0 0 0 0 0 0 0 0 0
40 0 0 0 0 0 0 0 0 0 0 0 0 0 755 0 0 0 0 0 0 0 0 0 0 0
16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 145 0 0 0 0 0 0 0 0 0 0
40 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 834 0 0 0 0 0 0 0 0 0
13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 160 0 0 0 0 0 0 0 0
319 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 634 0 0 0 0 0 0 0
230 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 760 0 0 0 0 0 0
18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 247 0 0 0 0 0
132 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 901 0 0 0 0
29 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 479 0 0 0
52 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1847 0 0
26 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 448 0
165 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 396
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 187

SVC (100 WORDS BY CODE)

TRAIN_R2_SCORE = 0.8660274721935209

TEST_R2_SCORE = 0.8574120221236249

BEST_PARAMS: [{'C': 10, 'KERNEL': 'LINEAR'}]

params {'kernel': ('linear', 'rbf'), 'C': [10, 20]}

X_train.shape - X_test.shape - len(y_train) - len(y_test)

(65812, 2700) - (16453, 2700) - 65812 - 16453

---

train_f1_score = [array([0.36168826, 0.66088117, 0.94627105, 0.99273608, 0.93363162,

0.9073154 , 0.89071038, 0.9119452 , 0.91848373, 0.90918919,

0.9622438 , 0.92756133, 0.87660327, 0.98651802, 0.98189068,

0.94857143, 0.97431555, 0.96634615, 0.79063803, 0.85341426,

0.96040987, 0.93725222, 0.98140127, 0.98680361, 0.96203209,

0.83623877, 1.])]

test_f1_score = [array([0.34432644, 0.66099291, 0.94719472, 0.99328859, 0.91975309,

0.90215827, 0.88489209, 0.9052751 , 0.91751085, 0.90574713,

0.90909091, 0.92363636, 0.85614647, 0.98293173, 0.96722408,

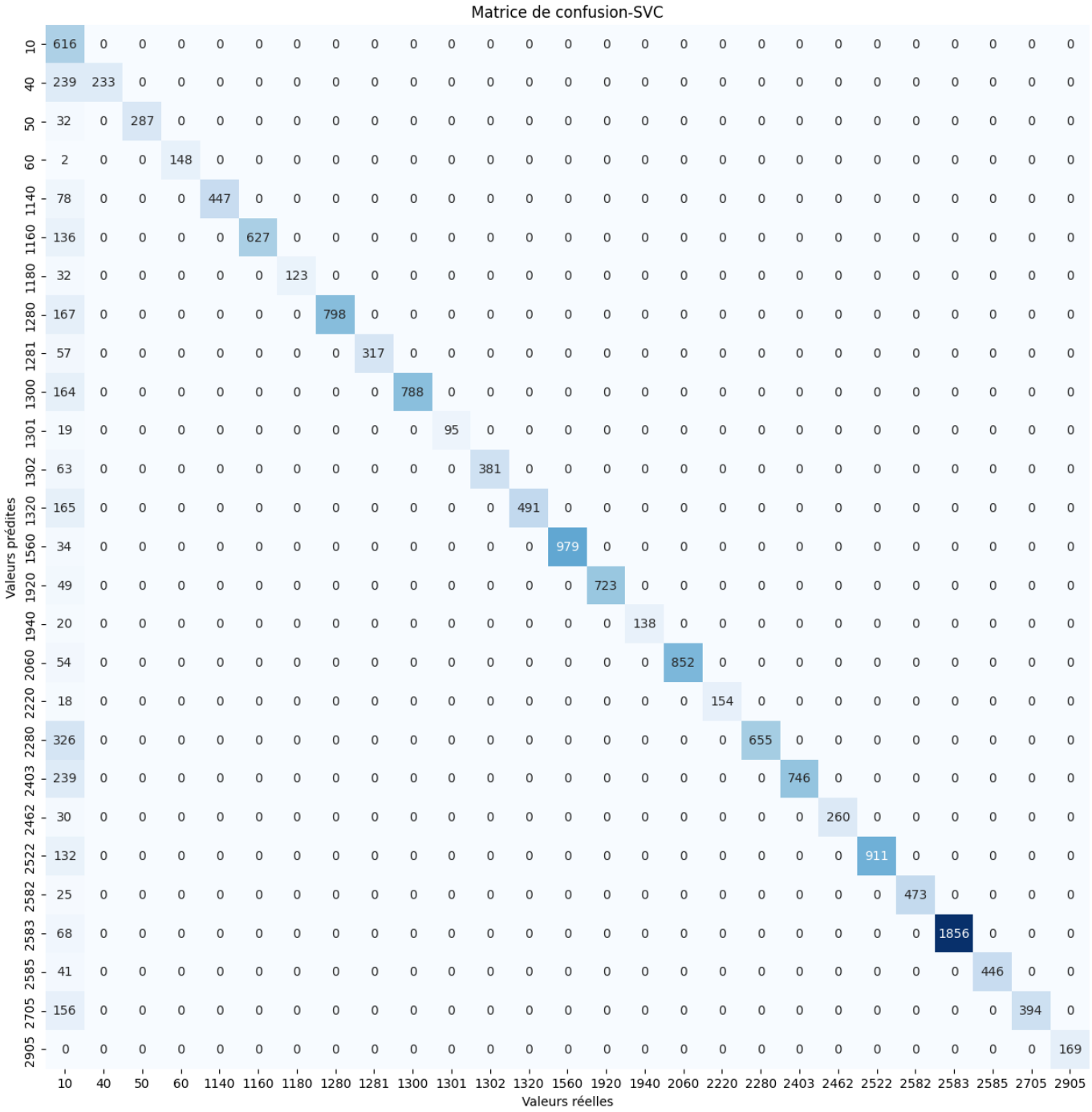
0.93243243, 0.96928328, 0.94478528, 0.8007335 , 0.86192952,

0.94545455, 0.93244626, 0.97425335, 0.98201058, 0.95605573,

0.83474576, 1.])]

train_mse_result = 455162.75148909015

test_mse_result = 492912.0065641524


Matrice de confusion-SVC
Valeurs prédites
Valeurs réelles
10 40 50 60 1140 1160 1180 1280 1281 1300 1301 1302 1320 1560 1920 1940 2060 2220 2280 2403 2462 2522 2582 2583 2585 2705 2905
10 616 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
40 239 233 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
50 32 0 287 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
60 2 0 0 148 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1140 78 0 0 0 447 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1160 136 0 0 0 0 627 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1180 32 0 0 0 0 0 123 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1280 167 0 0 0 0 0 0 798 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1281 57 0 0 0 0 0 0 0 317 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1300 164 0 0 0 0 0 0 0 0 788 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1301 19 0 0 0 0 0 0 0 0 0 95 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1302 63 0 0 0 0 0 0 0 0 0 0 381 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1320 165 0 0 0 0 0 0 0 0 0 0 0 491 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1560 34 0 0 0 0 0 0 0 0 0 0 0 0 979 0 0 0 0 0 0 0 0 0 0 0 0 0
1920 49 0 0 0 0 0 0 0 0 0 0 0 0 0 723 0 0 0 0 0 0 0 0 0 0 0 0
1940 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 138 0 0 0 0 0 0 0 0 0 0 0
2060 54 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 852 0 0 0 0 0 0 0 0 0 0
2220 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 154 0 0 0 0 0 0 0 0 0
2280 326 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 655 0 0 0 0 0 0 0 0
2403 239 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 746 0 0 0 0 0 0 0
2462 30 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 260 0 0 0 0 0 0
2522 132 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 911 0 0 0 0 0
2582 25 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 473 0 0 0 0
2583 68 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1856 0 0 0
2585 41 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 446 0 0
2705 156 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 394 0
2905 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 0 169

KNN (300 WORD BY CODE)

TRAIN_R2_SCORE = 0.9067799185558865

TEST_R2_SCORE = 0.9002613505135841

ESTIMATOR KNEIGHBORSCLASSIFIER()

PARAMS {'N_NEIGHBORS': [10]}

X_train.shape - X_test.shape - len(y_train) - len(y_test)

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

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

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,

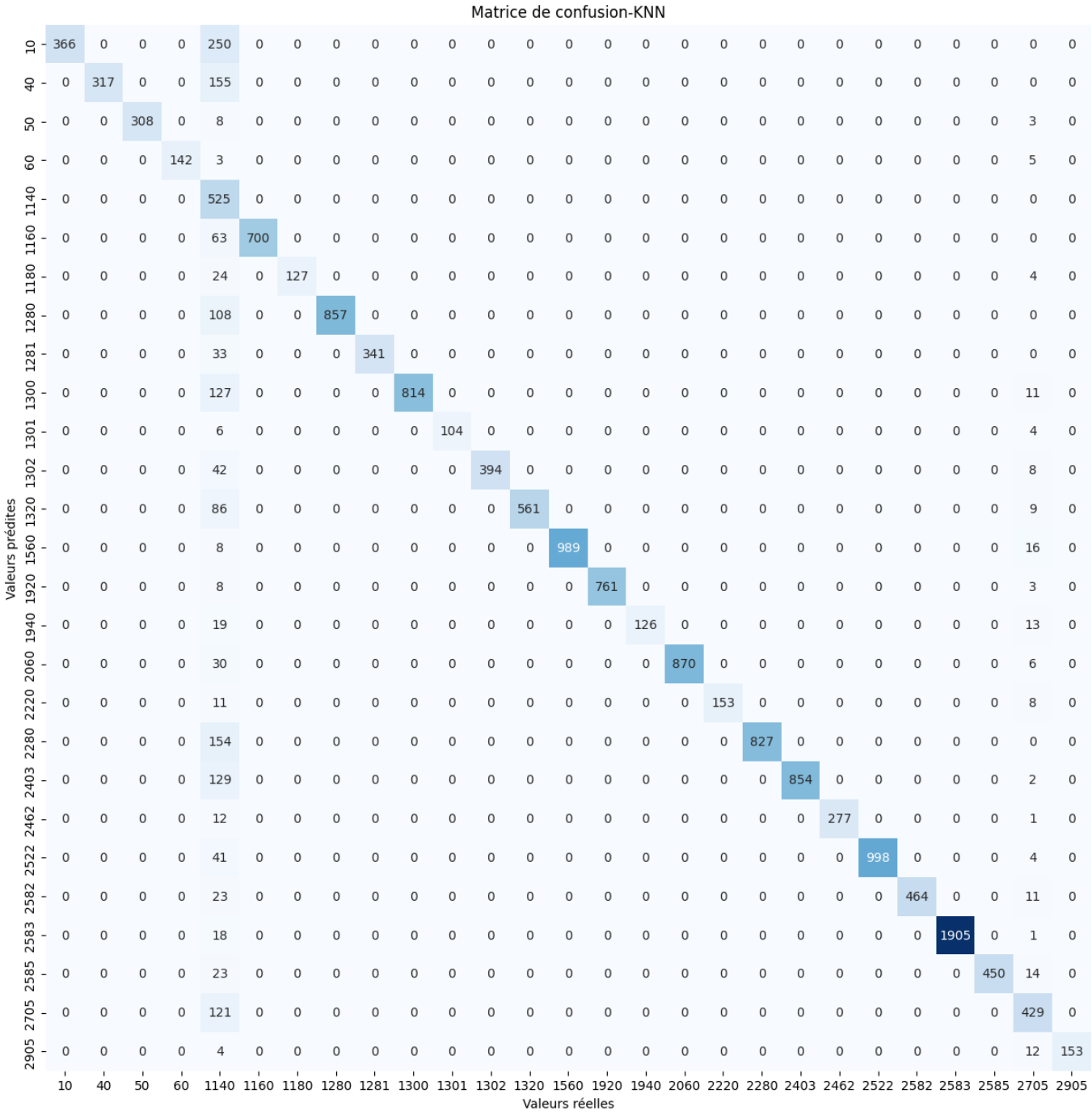
0.77019749, 0.95031056])]

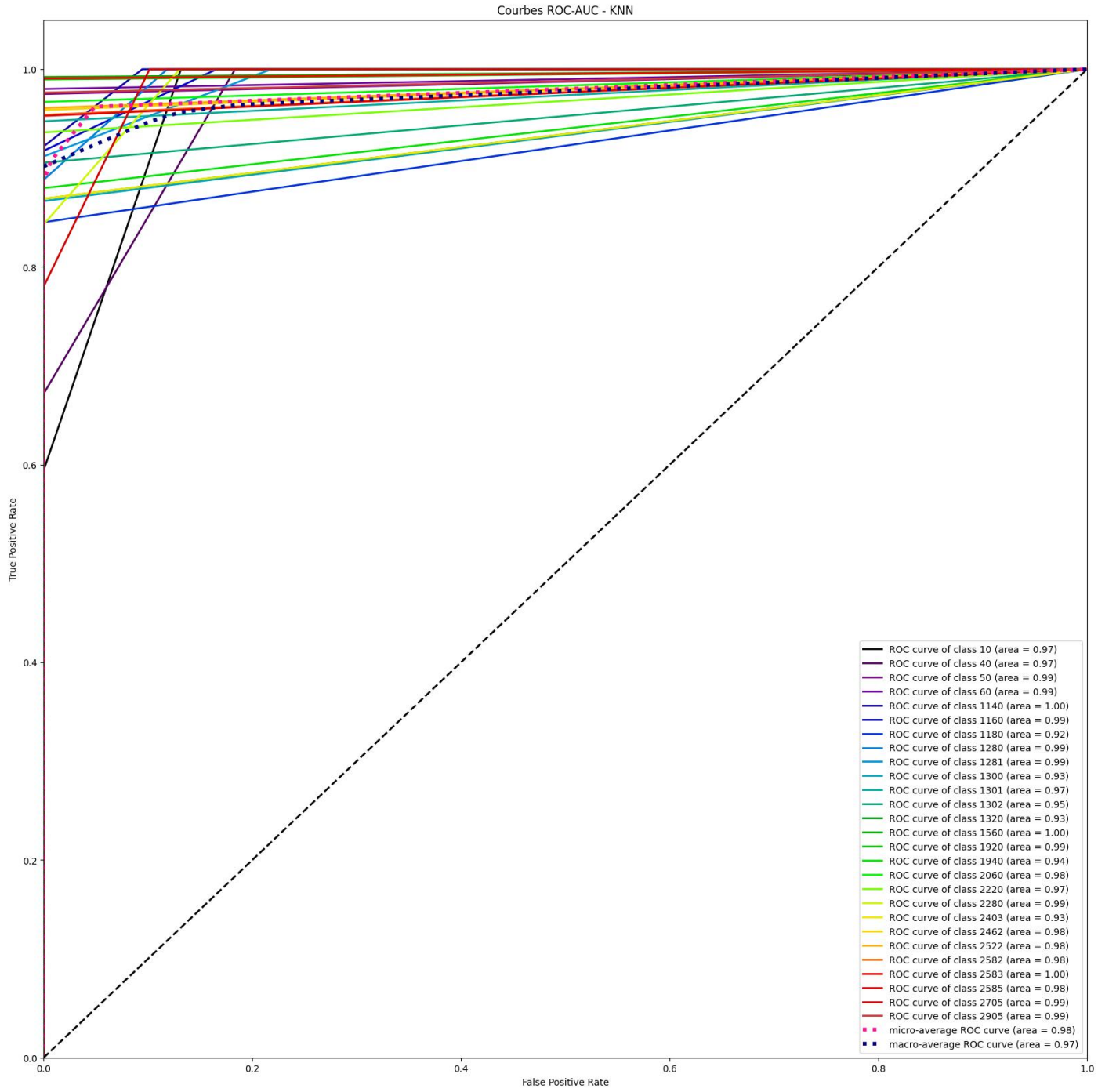
train_mse_result = 96349.62113292409

test_mse_result = 103203.23928766791

best_params: [{'n_neighbors': 10}]

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


Matrice de confusion-KNN
Valeurs prédites
Valeurs réelles
10 40 50 60 1140 1160 1180 1280 1281 1300 1301 1302 1320 1560 1920 1940 2060 2220 2280 2403 2462 2522 2582 2583 2585 2705 2905


Courbes ROC-AUC - KNN
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 10 (area = 0.97)
ROC curve of class 40 (area = 0.97)
ROC curve of class 50 (area = 0.99)
ROC curve of class 60 (area = 0.99)
ROC curve of class 1140 (area = 1.00)
ROC curve of class 1160 (area = 0.99)
ROC curve of class 1180 (area = 0.92)
ROC curve of class 1280 (area = 0.99)
ROC curve of class 1281 (area = 0.99)
ROC curve of class 1300 (area = 0.93)
ROC curve of class 1301 (area = 0.97)
ROC curve of class 1302 (area = 0.95)
ROC curve of class 1320 (area = 0.93)
ROC curve of class 1560 (area = 1.00)
ROC curve of class 1920 (area = 0.99)
ROC curve of class 1940 (area = 0.94)
ROC curve of class 2060 (area = 0.98)
ROC curve of class 2220 (area = 0.97)
ROC curve of class 2280 (area = 0.99)
ROC curve of class 2403 (area = 0.93)
ROC curve of class 2462 (area = 0.98)
ROC curve of class 2522 (area = 0.98)
ROC curve of class 2582 (area = 0.98)
ROC curve of class 2583 (area = 1.00)
ROC curve of class 2585 (area = 0.98)
ROC curve of class 2705 (area = 0.99)
ROC curve of class 2905 (area = 0.99)
micro-average ROC curve (area = 0.98)
macro-average ROC curve (area = 0.97)

NAIVE BAYES (300 WORDS BY CODE) – 30SEC TEMPS D'EXCUCUTION

ESTIMATOR MULTINOMIALNB()

PARAMS {'ALPHA': [1]}

TRAIN_R2_SCORE = 0.9078283595696833

TEST_R2_SCORE = 0.9073117364614356

MEAN_TRAIN_F1_SCORE= 0.9392493667037067

MEAN_TEST_F1_SCORE= 0.9379498799117439

train_mse_result = 229948.6231842217

test_mse_result = 236261.5843919042

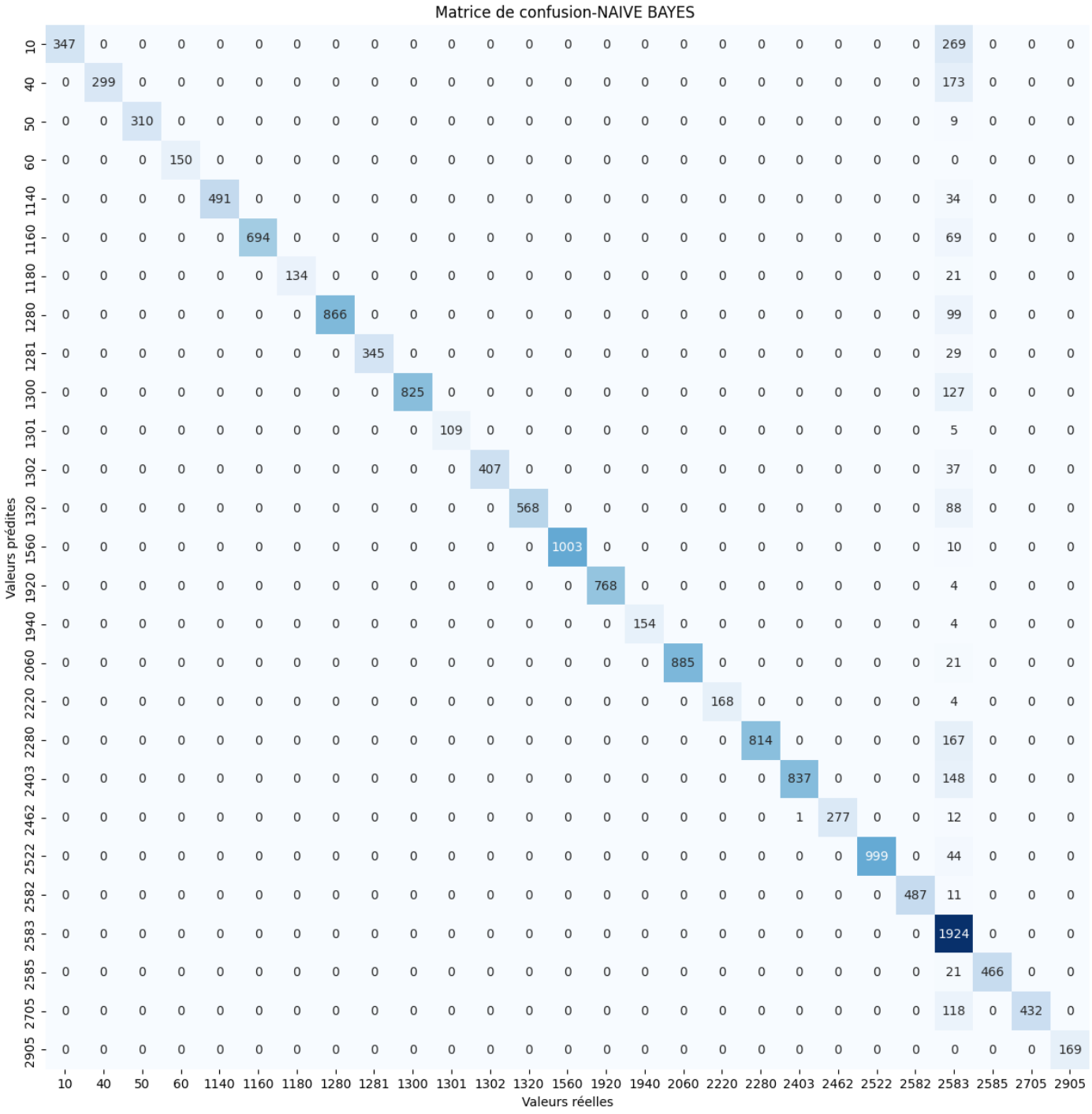
best_params: [{'alpha': 1}]

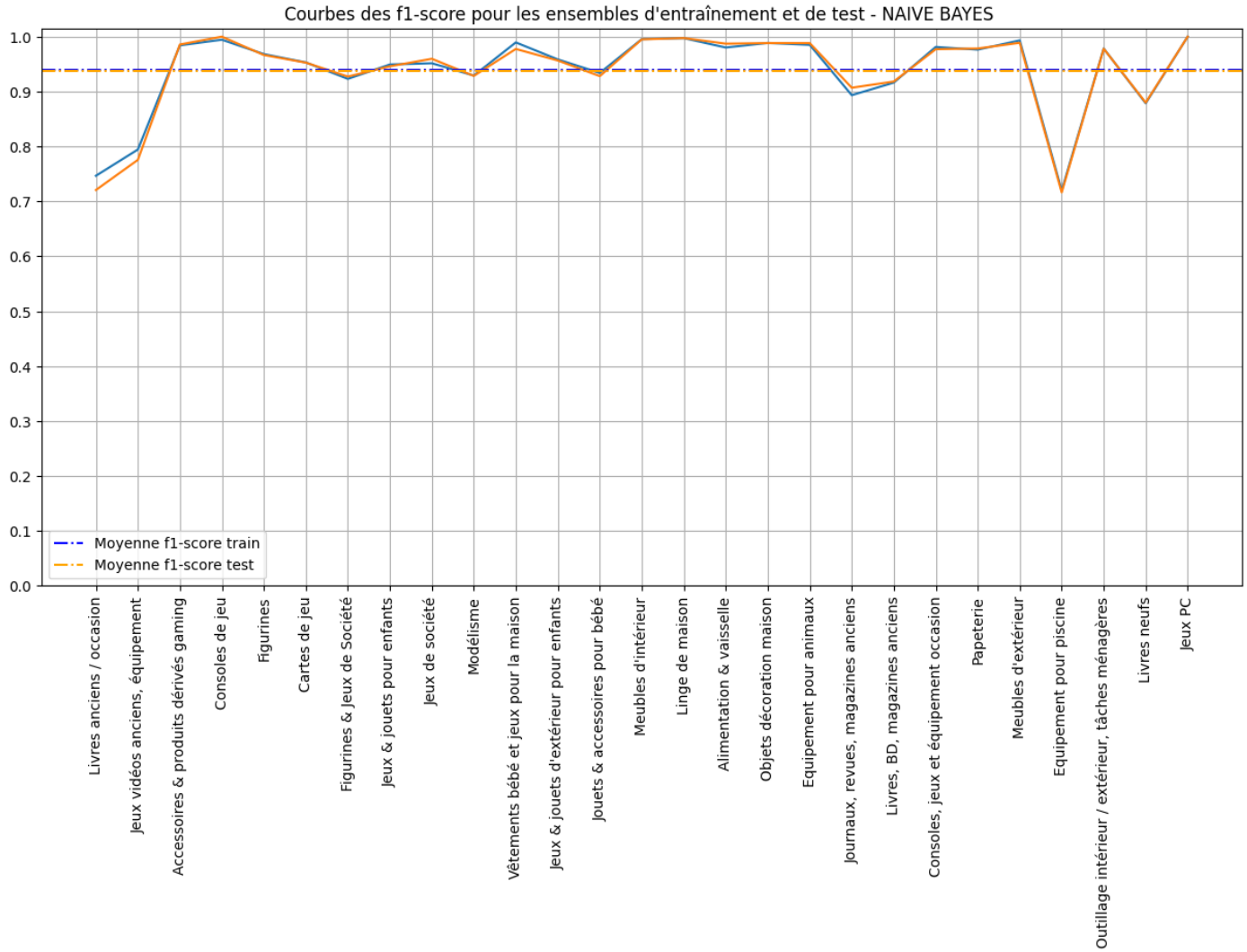
train_f1_score = [array([0.74661315, 0.79447115, 0.98415153, 0.99435939, 0.96825397,
0.95305318, 0.92307692, 0.94928335, 0.95154472, 0.92915893,
0.9894958 , 0.95852018, 0.93389297, 0.99550302, 0.99721813,
0.98020586, 0.98858892, 0.98505114, 0.89335485, 0.9162604 ,
0.98128708, 0.97629708, 0.99320071, 0.71987437, 0.97842105,
0.87859506, 1.])]

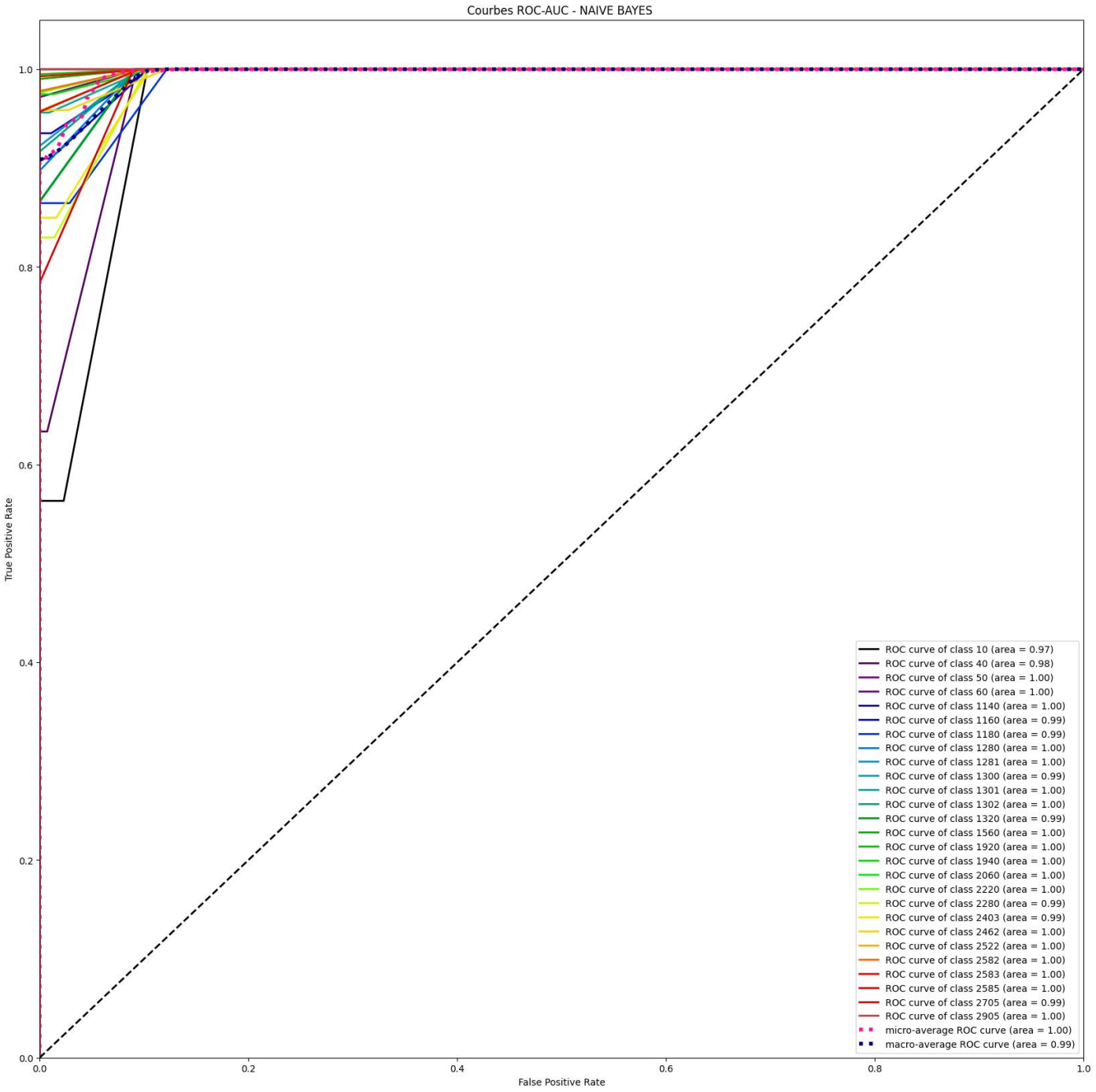
test_f1_score = [array([0.72066459, 0.77561608, 0.98569157, 1. , 0.96653543,
0.95264242, 0.92733564, 0.94593119, 0.9596662 , 0.92853123,
0.97757848, 0.95652174, 0.92810458, 0.99503968, 0.9974026 ,
0.98717949, 0.98827471, 0.98823529, 0.90696379, 0.91826659,
0.97707231, 0.9784525 , 0.98883249, 0.71630678, 0.97796432,
0.87983707, 1.])]

mean_train_f1_score= 0.9392493667037067

mean_test_f1_score= 0.9379498799117439


Matrice de confusion-NAIVE BAYES
Valeurs prédites
Valeurs réelles
10 40 50 60 1140 1160 1180 1280 1281 1300 1301 1302 1320 1560 1920 1940 2060 2220 2280 2403 2462 2522 2582 2583 2585 2705 2905


Courbes des f1-score pour les ensembles d'entraînement et de test - NAIVE BAYES
Moyenne f1-score train
Moyenne f1-score test
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Livres anciens / occasion
Jeux vidéos anciens, équipement
Accessoires & produits dérivés gaming
Consoles de jeu
Figurines
Cartes de jeu
Figurines & Jeux de Société
Jeux & jouets pour enfants
Jeux de société
Modélisme
Vêtements bébé et jeux pour la maison
Jeux & jouets d'extérieur pour enfants
Jouets & accessoires pour bébé
Meubles d'intérieur
Linge de maison
Alimentation & vaisselle
Objets décoration maison
Equipement pour animaux
Journaux, revues, magazines anciens
Livres, BD, magazines anciens
Consoles, jeux et équipement occasion
Papeterie
Meubles d'extérieur
Equipement pour piscine
Outillage intérieur / extérieur, tâches ménagères
Livres neufs
Jeux PC



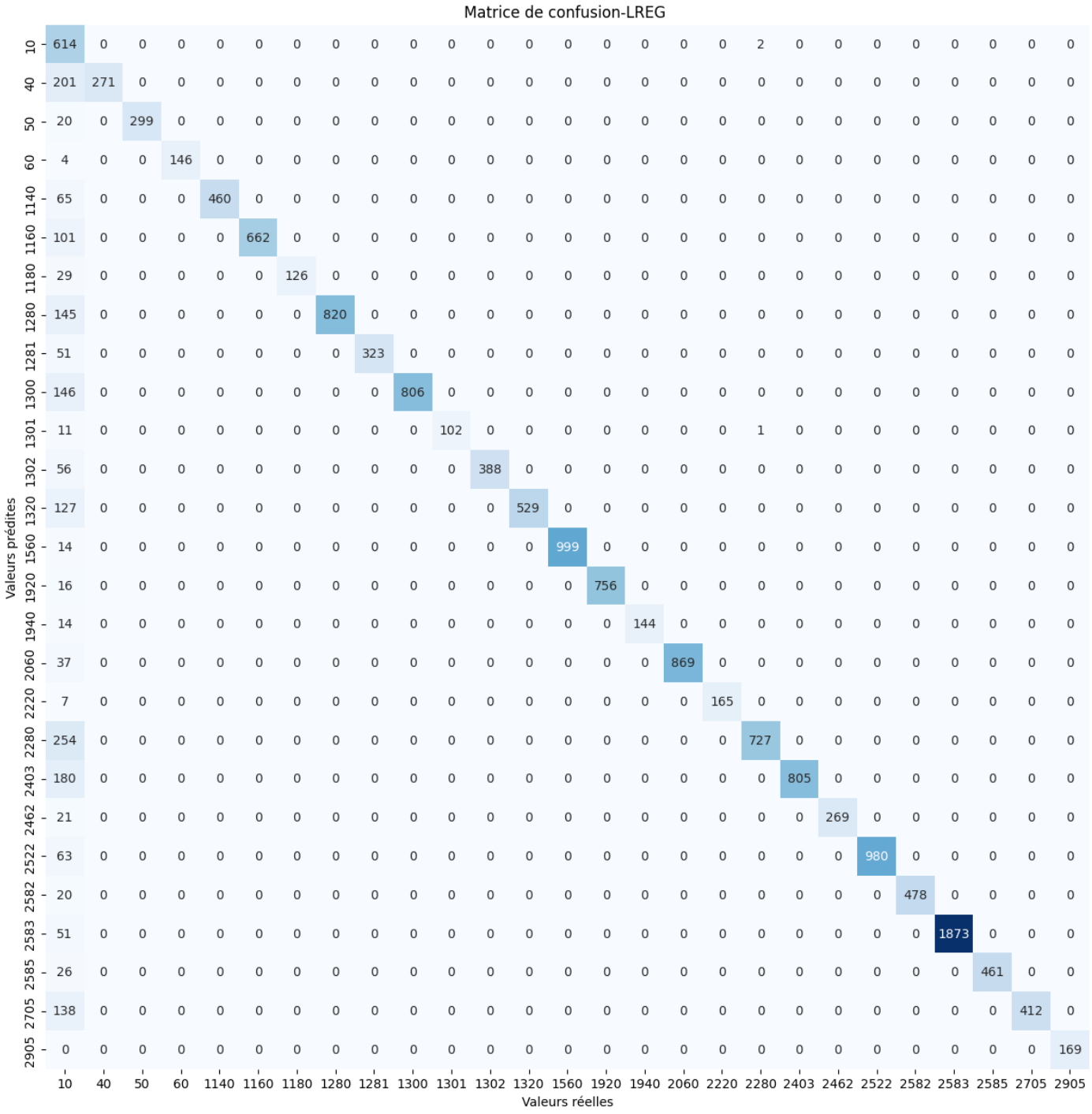
LREG (300 WORDS BY CODE)

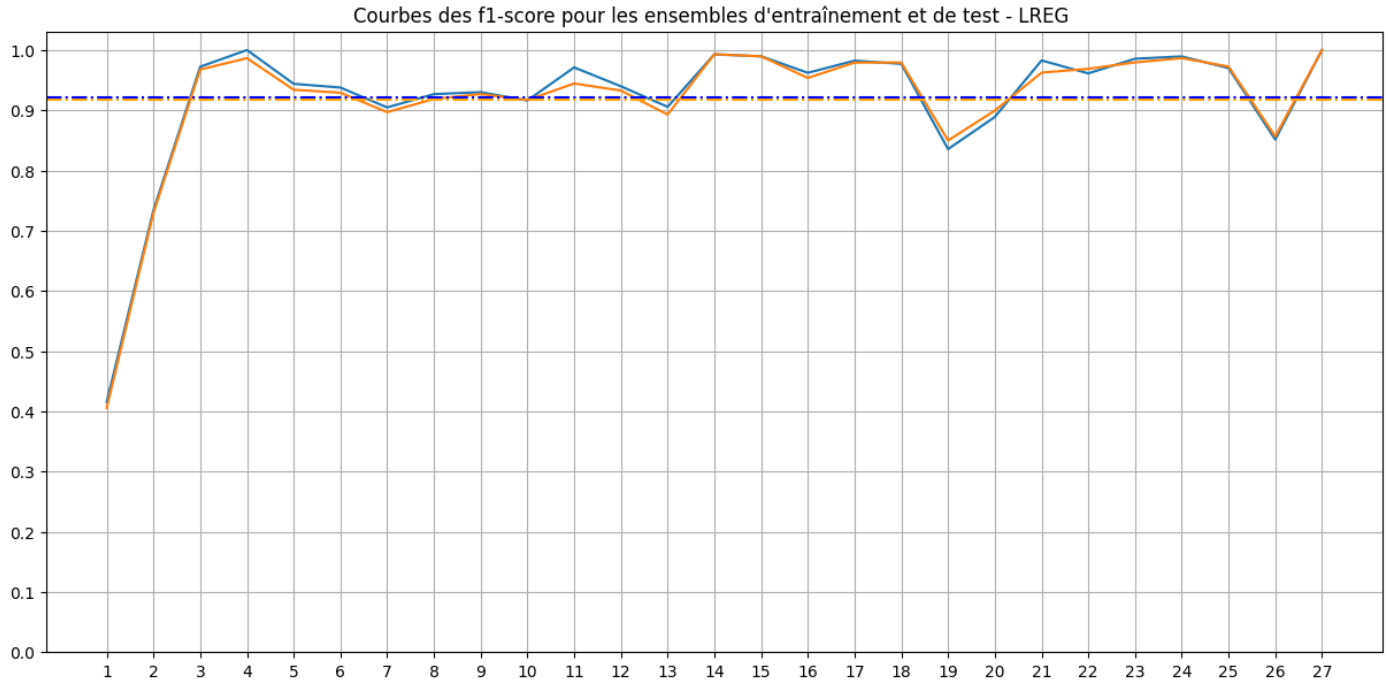
ESTIMATOR LOGISTICREGRESSION()

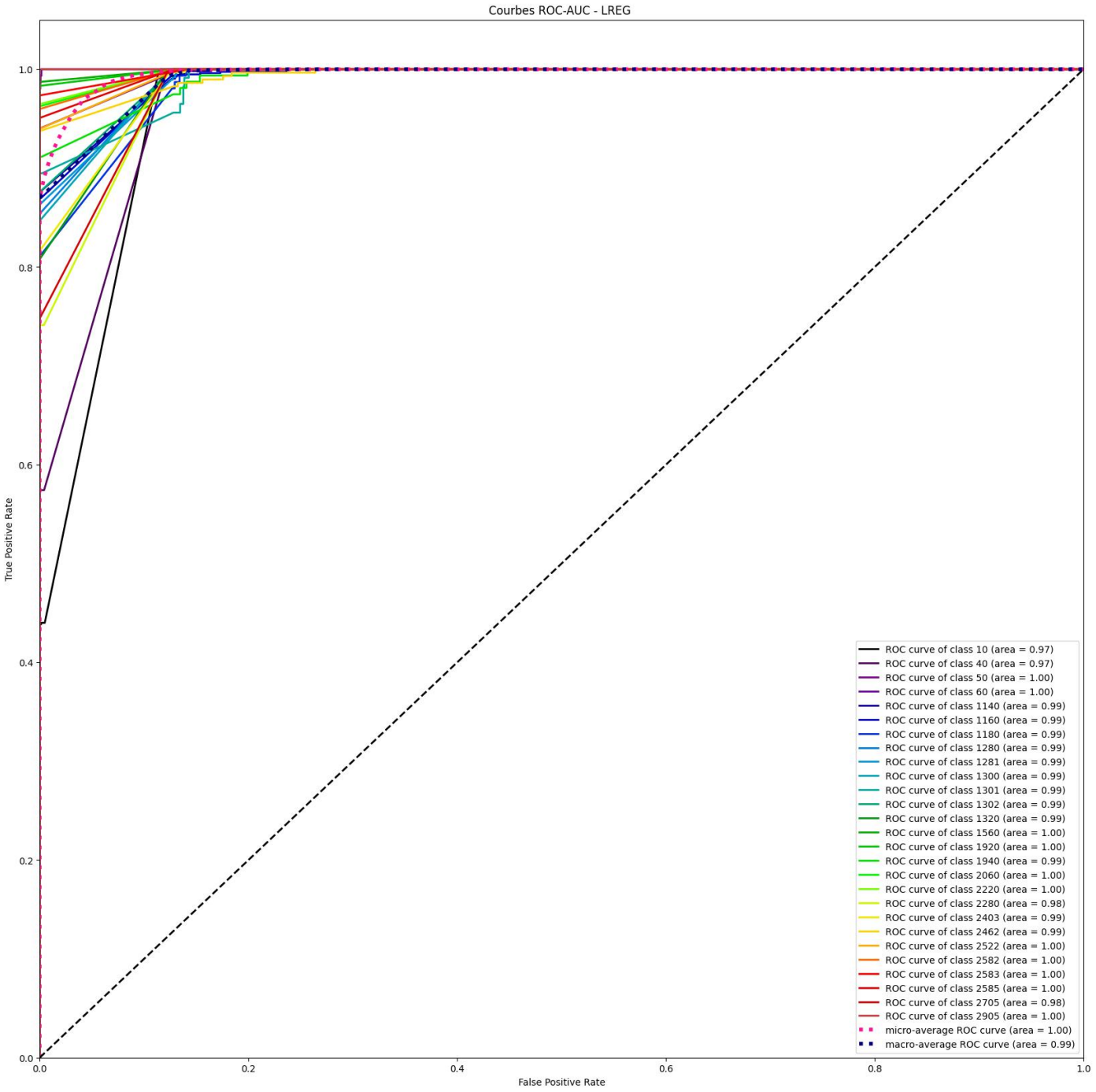
PARAMS {'C': [50]}

TRAIN_R2_SCORE = 0.8932109645657327

TEST_R2_SCORE = 0.8905974594298912


Matrice de confusion-LREG
Valeurs prédites
Valeurs réelles
10 40 50 60 1140 1160 1180 1280 1281 1300 1301 1302 1320 1560 1920 1940 2060 2220 2280 2403 2462 2522 2582 2583 2585 2705 2905
614 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0
201 271 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
20 0 299 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 146 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
65 0 0 0 460 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
101 0 0 0 0 662 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
29 0 0 0 0 0 126 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
145 0 0 0 0 0 0 820 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
51 0 0 0 0 0 0 0 323 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
146 0 0 0 0 0 0 0 0 806 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 102 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
56 0 0 0 0 0 0 0 0 0 0 388 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
127 0 0 0 0 0 0 0 0 0 0 0 529 0 0 0 0 0 0 0 0 0 0 0 0 0 0
14 0 0 0 0 0 0 0 0 0 0 0 0 999 0 0 0 0 0 0 0 0 0 0 0 0 0
16 0 0 0 0 0 0 0 0 0 0 0 0 0 756 0 0 0 0 0 0 0 0 0 0 0 0
14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 144 0 0 0 0 0 0 0 0 0 0 0
37 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 869 0 0 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 165 0 0 0 0 0 0 0 0 0
254 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 727 0 0 0 0 0 0 0 0
180 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 805 0 0 0 0 0 0 0
21 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 269 0 0 0 0 0 0
63 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 980 0 0 0 0 0
20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 478 0 0 0 0
51 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1873 0 0 0
26 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 461 0 0
138 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 412 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 0 169


Courbes des f1-score pour les ensembles d'entraînement et de test - LREG
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27



{'mean_fit_time': array([108.86818051]),
'std_fit_time': array([3.84594668]),
'mean_score_time': array([0.80805755]),
'std_score_time': array([0.09732477]),
'param_C': masked_array(data=[50],
mask=[False],
fill_value='?',
dtype=object),
'params': [{'C': 50}],
'split0_test_score': array([0.88809372]),
'split1_test_score': array([0.88872681]),

'split2_test_score': array([0.88412272]),

'mean_test_score': array([0.88698108]),

'std_test_score': array([0.00203763]),

'rank_test_score': array([1])}