Machine Learning & Application

2021-2022

Projet Rubik's cube

Le projet porte sur un problème d'apprentissage supervisé. Le problème fait parti des données du challenge des données ENS et s'intitule "Solve 2x2x2 Rubik's cube" et est présenté par la société LumenAI. Une vidéo décrivant le problème se trouve sur le site du collège de France.

Les autres challenges sont aussi très intéressants, mais nécessitent plus de connaissances en machine learning (par exemple de l'apprentissage sur des séries temporelles, sur des images, des sons, du texte, etc...). D'où le choix de ce challenge dont les données sont très proches de problèmes sur lesquels on a travaillé.

La résolution d'un rubik's cube peut être vu comme un problème d'intelligence artificielle (par exemple en utilisant des techniques de recherches avec des heuristiques). On peut même étudier le graphe du jeu du point de vue de la théorie des graphes et découvrir qu'en fait il existe toujours un chemin relativement court à une solution.

ex dans la litérature:

- "The Diameter of the Rubik's Cube Group Is Twenty", *T. Rokicki, H. Kociemba, M. Davidson, and J. Dethridge*, SIAM Review, 2014, Vol. 56, No. 4 : pp. 645-670.
- "Solving Rubik's Cube Using Graph Theory", Khemani C., Doshi J., Duseja J., Shah K., Udmale S., Sambhe V. (2019) in: Verma N., Ghosh A. (eds) Computational Intelligence: Theories, Applications and Future Directions - Volume I. Advances in Intelligent Systems and Computing, vol 798. Springer, Singapore.

Ici, on a une base de données qui contient la description de rubik's cubes ainsi que le nombre minimal de coups pour les résoudre. On ne sait pas comment les problèmes ont été générés (est-ce que cela entraine un biais dans les problèmes, est-ce que plusieurs problèmes similaires sont présents dans la base? (Ici par similaire, on pourrait peut être avoir deux problèmes qui apparaissent dans la base, mais en permuttant certaines couleurs, on aurait peut-être exactement le même problème!)). Cependant, on vous demande de constuire un modèle pour nous aider à prédire le nombre de coups minimal pour un problème donné. Ensuite, vous pourriez utiliser ce modèle dans un algorithme de recherche étudié en cours d'IA.

On peut le voir comme un problème de regression où il faut deviner le nombre minimal de coups pour résoudre le rubik's cube, ou comme un problème de classification où la classe d'un état du rubik's cube est le nombre minimal de coups pour le résoudre (donc on pourrait avoir au plus 19 classes). Toutes les méthodes que l'on a vu en cours peuvent s'appliquer.

Les données et le site du challenge

Le projet s'effectue en binôme. Vous devez ouvrir un compte pour le binôme sur le site du challenge, choisissez de participer seul (le binôme sera un seul participant au challenge), puis inscrivez-vous au challenge du cours *M1 MIAGE Dauphine - PSL - 2021-2022*.

Vous aurez accès à trois ensembles:

- `x_train` qui contient la description de 1.837.079 différents rubik's cubes. Chaque rubik's cube est représenté par 25 attributs (lisez la description sur le site du challenge).
- `y_train` qui contient le nombre minimal de coups pour résoudre chacun des 1.837.079 différents rubik's cubes. Ces données sont vos données d'entrainement.
- enfin `x_test` qui contient la description de 1.837.080 nouveaux rubik's cubes. Vous ne connaissez pas le nombre minimal pour chacun de ces problèmes.

Pour participer aux challenges, il vous faudra uploader sur le site votre prédiction sur les rubik's cubes du fichier `x_test` et le site du challenge vous donnera un score. Pour ce score, le site utilise l'erreur moyenne absolue: pour les n=1.837.080 exemples du fichier test, on fait la moyenne entre le vrai nombre minimal de coups y_i et votre prédiction z_i:

$$\frac{\sum_{i=1}^{n} |y_i - z_i|}{n}$$

Malheureusement (pour vous), le site ne vous donnera pas plus d'information que votre score, vous ne pourrez pas savoir quelles sont vos bonnes prédictions et quelles sont vos erreurs. Pire, le site vous permettra d'uploader une prédiction que deux fois par jour!

Soumission et rappot

Un des membres du binôme devra remplir le formulaire Forms de l'équipe Teams du cours (onglet General) pour enregistrer les membres du binôme et le login du binôme qui utilisé sur le site du challenge ENS. Avant le **jeudi 2 décembre à 12h** vous devez 1) avoir créé votre compte pour le binôme et inscrit le binôme sur le challenge du cours, et 2) rempli les informations sur le formulaire Forms.

Vous pouvez faire le projet seul, mais vous serez évalué comme un binôme. Si vous tenez absolument à former un trinôme, contactez moi par email, mais sachez que dès lors, les attentes seront plus élevées.

Le deadline pour le projet sera le **dimanche 20 décembre 23:59** Vous devrez à ce moment là avoir fait trois choses:

- avoir rendu un rapport
- avoir rendu un notebook jupyter ou collab contenant le code pour générer votre solution
- avoir soumis une solution sur le site du challenge ENS.

Le notebook et le rapport seront à soumettre sous myCourse.

Le rapport est un document **pdf** et devra être un document structuré qui explique vos choix, explique votre solution et donne votre résultat. Ne présentez ni le cours, ni le contexte, seul

votre travail est important. Le rapport est de **6** pages maximum au format A4 (sans utiliser une taille de police inférieure strictement à 12). Vous pouvez ajouter une annexe à ce rapport (au format pdf ou sous la forme d'un notebook jupyter), étant entendu que le lecteur n'est pas obligé de lire l'annexe. Votre mission est de proposer un modèle de prédiction pour ce problème, votre rapport doit justifier comment vous avez répondu (complètement ou pas) à cette mission (par exemple, vous pouvez décrire ce que vous avez essayé, ce qui a marché ou non, pourquoi vous avez essayé autre chose...). Une autre façon de décrire ce qu'on attend du rapport est la suivante: votre manageur a donné à plusieurs équipes la même tâche d'apprentissage supervisé. Vous devez lui présenter dans ce rapport des arguments qui justifient la qualité de votre approche et de vos résultats. Votre manageur connait le problème, mais n'est pas forcément un expert du domaine. A vous de le convaincre d'utiliser votre solution! (attention, si vous connaissez aussi les limitations de votre solution, il est bon de les exposer aussi!).

L'évaluation portera sur la qualité de votre analyse, même si vos résultats sont peu concluant. Pour caricaturer, un modèle qui gagnerait la compétition sans pouvoir expliquer ce qu'il a fait n'aura pas une bonne note pour le projet du cours (mais bravo, il a gagné la compétition!). Autre caricature, un projet qui applique un seul algorithme et conclue que ça ne fonctionne pas bien n'aura pas non plus une bonne note.

Une soutenance sera organisée lors de la première semaine de cours (après les examens). Elle permettra de compléter l'évaluation et de vous donner un retour sur votre travail. Cette soutenance ne demande aucune préparation de votre part. Elle durera une douzaine/quinzaine de minutes par groupe. La soutenance consistera en un échange au sujet de vos résultats et votre rapport. Si la soutenance fait apparaître qu'un des membres n'a pas beaucoup contribué, sa note pourra être revue à la baisse. Egalement, on pourra vous demander de montrer le code et de fournir les résultats que vous avez obtenu lors des exercices d'implémentation des TDs.

Ce projet compte pour 40% de la note de l'UE. Il est donc souhaitable que la note corresponde au travail de votre groupe, et non aux conseils d'autres groupes, d'autres étudiants ou d'internet. Si vous utilisez des sources (articles de recherche, posts sur internet, etc...), vous devez mentionnez vos sources dans le rapport (sinon, cela s'appelle du plagiat, et cela peut être puni par un conseil de discipline).

Les quelques lignes de code ci-dessous lisent simplement les fichiers sources et affiche la taille des données.

In [141…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


dfX = pd.read_csv("train_input.csv",index_col=0)
dfY = pd.read_csv("train_output.csv",index_col=0)
X = dfX.to_numpy()
y = dfY['distance'].values
print("taille des données:")
print("X:", X.shape)
print("y:", np.size(y))
```

```
dfT = pd.read_csv("test_input.csv",index_col=0)
Xtest = dfT.to_numpy()
print("test:", Xtest.shape)
```

```
/Users/salimzerhouni/opt/anaconda3/lib/python3.9/site-packages/numpy/lib/array
setops.py:583: FutureWarning: elementwise comparison failed; returning scalar
instead, but in the future will perform elementwise comparison
  mask |= (ar1 == a)
taille des données:
X: (1837079, 24)
y: 1837079
test: (1837080, 24)
```

In [264…

```
print("Il y a ", X.size, " observations avec ", X[0].size, " attributs")
```

```
Il y a  44089896  observations avec  24  attributs
```

In [265…

```
dfX
```

Out[265…

ID	pos0	pos1	pos2	pos3	pos4	pos5	pos6	pos7	pos8	pos9	...	pos14	pos15
0	4	1	1	1	6	2	6	6	5	4	...	4	4
1	6	5	2	1	2	2	6	3	4	4	...	4	1
2	5	3	3	2	3	1	6	5	1	1	...	4	6
3	5	5	4	1	2	1	6	1	2	2	...	4	4
4	4	2	1	5	1	3	6	6	3	3	...	4	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1837074	2	1	3	3	5	3	6	6	1	5	...	4	3
1837075	2	3	3	5	6	4	6	1	3	1	...	4	5
1837076	3	3	3	2	2	4	6	6	1	4	...	4	3
1837077	5	3	5	1	5	3	6	3	6	4	...	4	4
1837078	3	1	4	2	1	5	6	4	6	2	...	4	1

1837079 rows × 24 columns

In [266…

```
dfY
```

Out[266…

ID	distance
0	11
1	11
2	11
3	9
4	12
...	...
1837074	11

ID	distance
1837075	9
1837076	12
1837077	11
1837078	11

1837079 rows × 1 columns

Première approche

In [276…
```
#DecisionTree
from sklearn.tree import DecisionTreeClassifier
from math import sqrt
from sklearn.metrics import precision_score
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10)
dtree = DecisionTreeClassifier(criterion="gini",max_depth=3)
dtree.fit(X_train, y_train)
ydt = dtree.predict(X_test)
mean_absolute_error(y_test,ydt)
```

Out[276…
```
0.857992030831537
```

In [277…
```
np.bincount(ydt)
```

Out[277…
```
array([     0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0, 183708])
```

In [85]:
```
dfY.value_counts()
```

Out[85]:
```
distance
11          675426
10          465294
12          391268
9           180254
8            57074
13           45140
7            16529
6             4485
5             1128
4              267
14             138
3               60
2               13
1                3
dtype: int64
```

In [7]:
```
#NaïveBayes
from sklearn.naive_bayes import GaussianNB
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10)
gnb = GaussianNB()
```

```
gnb.fit(X_train,y_train)
ydt2 = gnb.predict(X_test)
mean_absolute_error(y_test,ydt2)
```

```
1.0050188342369413
```

Des cubes similaires ?

Programme de rotation du cube

In [9]:
```
import itertools
Xn = []
for i in range(6):
    for j in range(i+1,6):
        for k in range(1,4):
            Xn1 = []
            Xn1.append(np.roll(X[0,:4*i],4*k).tolist())
            Xn1.append(np.roll(X[0,4*i:4*(i+1)],1*k).tolist())
            Xn1.append(np.roll(X[0,4*(i+1):4*j],4*k).tolist())
            Xn1.append(np.roll(X[0,4*j:4*(j+1)],1*k).tolist())
            Xn1.append(np.roll(X[0,4*(j+1):],4*k).tolist())
        flat_list = itertools.chain(*Xn1)
        Xn.append(list(flat_list))
Xn
```

Out[9]:
```
[[1, 1, 1, 4, 2, 6, 6, 6, 5, 6, 4, 4, 1, 3, 3, 5, 2, 3, 5, 3, 5, 4, 2, 2],
 [1, 1, 1, 4, 6, 2, 6, 6, 4, 2, 2, 5, 5, 6, 4, 4, 1, 3, 3, 5, 2, 3, 5, 3],
 [1, 1, 1, 4, 5, 4, 2, 2, 6, 2, 6, 6, 6, 4, 4, 5, 2, 3, 5, 3, 1, 3, 3, 5],
 [1, 1, 1, 4, 6, 2, 6, 6, 5, 4, 2, 2, 5, 6, 4, 4, 3, 3, 5, 1, 2, 3, 5, 3],
 [1, 1, 1, 4, 5, 4, 2, 2, 5, 6, 4, 4, 1, 3, 3, 5, 6, 2, 6, 6, 3, 5, 3, 2],
 [4, 1, 1, 1, 2, 6, 6, 6, 4, 2, 2, 5, 5, 6, 4, 4, 1, 3, 3, 5, 2, 3, 5, 3],
 [4, 1, 1, 1, 2, 6, 6, 6, 5, 4, 2, 2, 6, 4, 4, 5, 2, 3, 5, 3, 1, 3, 3, 5],
 [4, 1, 1, 1, 2, 6, 6, 6, 5, 6, 4, 4, 5, 4, 2, 2, 3, 3, 5, 1, 2, 3, 5, 3],
 [4, 1, 1, 1, 2, 6, 6, 6, 5, 4, 2, 2, 5, 6, 4, 4, 1, 3, 3, 5, 3, 5, 3, 2],
 [6, 2, 6, 6, 4, 1, 1, 1, 4, 2, 2, 5, 6, 4, 4, 5, 2, 3, 5, 3, 1, 3, 3, 5],
 [6, 2, 6, 6, 4, 1, 1, 1, 4, 2, 2, 5, 5, 6, 4, 4, 3, 3, 5, 1, 2, 3, 5, 3],
 [6, 2, 6, 6, 4, 1, 1, 1, 4, 2, 2, 5, 1, 3, 3, 5, 5, 6, 4, 4, 3, 5, 3, 2],
 [4, 1, 1, 1, 6, 2, 6, 6, 5, 4, 2, 2, 6, 4, 4, 5, 3, 3, 5, 1, 2, 3, 5, 3],
 [4, 1, 1, 1, 6, 2, 6, 6, 5, 4, 2, 2, 6, 4, 4, 5, 1, 3, 3, 5, 3, 5, 3, 2],
 [6, 2, 6, 6, 5, 4, 2, 2, 5, 6, 4, 4, 4, 1, 1, 1, 3, 3, 5, 1, 3, 5, 3, 2]]
```

In [10]:
```
for i in range(len(Xn)):
    if Xn[i] in X.tolist():
        print("YES")
```

Pas de "faux" doublons.

Réarrangement des combinaisons

In [11]:
```
a = np.where(y==1)
a
```

Out[11]:
```
(array([ 365030,  411427, 1266419]),)
```

Recherche de cubes à distance 1

In [12]:
```
print(X[365030],X[411427],X[1266419])
np.split(X[1266419],6)
```

```
[5 1 6 3 5 1 6 3 2 4 4 2 2 4 4 2 1 3 3 1 6 5 5 6] [1 1 6 6 1 1 6 6 3 5 4 2 3 5
4 2 4 4 3 3 2 2 5 5] [2 2 4 4 1 1 6 6 6 1 1 6 2 4 4 2 3 3 3 3 5 5 5 5]
```

Out[12]:
```
[array([2, 2, 4, 4]),
 array([1, 1, 6, 6]),
 array([6, 1, 1, 6]),
 array([2, 4, 4, 2]),
 array([3, 3, 3, 3]),
 array([5, 5, 5, 5])]
```

Fonction de réindexation des cubes

In [3]:
```
def faceCube(dfX):
    df = pd.DataFrame()
    df['pos0'] = dfX['pos18']
    df['pos1'] = dfX['pos17']
    df['pos2'] = dfX['pos19']
    df['pos3'] = dfX['pos16']
    df['pos4'] = dfX['pos21']
    df['pos5'] = dfX['pos22']
    df['pos6'] = dfX['pos20']
    df['pos7'] = dfX['pos23']
    df['pos8'] = dfX['pos1']
    df['pos9'] = dfX['pos5']
    df['pos10'] = dfX['pos0']
    df['pos11'] = dfX['pos4']
    df['pos12'] = dfX['pos2']
    df['pos13'] = dfX['pos6']
    df['pos14'] = dfX['pos3']
    df['pos15'] = dfX['pos7']
    df['pos16'] = dfX['pos14']
    df['pos17'] = dfX['pos13']
    df['pos18'] = dfX['pos10']
    df['pos19'] = dfX['pos9']
    df['pos20'] = dfX['pos11']
    df['pos21'] = dfX['pos8']
    df['pos22'] = dfX['pos15']
    df['pos23'] = dfX['pos12']
    return df
```

In [569…
```
dfX2 = faceCube(dfX2,dfX)
dfX2
```

Out[569…

ID	pos0	pos1	pos2	pos3	pos4	pos5	pos6	pos7	pos8	pos9	...	pos14	pos15
0	3	3	5	1	3	5	2	3	1	2	...	1	6
1	5	1	3	3	3	5	5	2	5	2	...	1	3
2	4	4	3	4	2	5	2	2	3	1	...	2	5
3	6	6	3	1	3	5	6	5	5	1	...	1	1
4	2	6	1	6	1	5	2	5	2	3	...	5	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1837074	2	4	6	5	1	5	6	2	1	3	...	3	6
1837075	6	4	1	1	6	5	2	2	3	4	...	5	1
1837076	1	6	5	2	1	5	1	2	3	4	...	2	6
1837077	4	6	2	2	6	5	2	1	3	3	...	1	3

ID	pos0	pos1	pos2	pos3	pos4	pos5	pos6	pos7	pos8	pos9	...	pos14	pos15
1837078	1	5	6	4	2	5	2	3	1	5	...	2	4

1837079 rows × 24 columns

In [286…
```
X2 = dfX2.to_numpy()
```

Ajout des attributs "nombre de couleur par face"

In [90]:
```
f=[]
b=[]
l=[]
r=[]
u=[]
d=[]

for i in range(len(dfX2['pos0'])):

    f.append(len(np.unique([dfX2['pos0'][i],dfX2['pos1'][i],dfX2['pos2'][i],d
    b.append(len(np.unique([dfX2['pos4'][i],dfX2['pos5'][i],dfX2['pos6'][i],d
    l.append(len(np.unique([dfX2['pos8'][i],dfX2['pos9'][i],dfX['pos10'][i],d
    r.append(len(np.unique([dfX2['pos12'][i],dfX2['pos13'][i],dfX2['pos14'][i
    u.append(len(np.unique([dfX2['pos16'][i],dfX2['pos17'][i],dfX2['pos18'][i
    d.append(len(np.unique([dfX2['pos20'][i],dfX2['pos21'][i],dfX2['pos22'][i
```

In [178…
```
dfX2['f'] = f
dfX2['b'] = b
dfX2['l'] = l
dfX2['r'] = r
dfX2['u'] = u
dfX2['d'] = d
dfX2
```

Out[178…

	pos0	pos1	pos2	pos3	pos4	pos5	pos6	pos7	pos8	pos9	...	pos20	pos21
0	3	3	5	1	3	5	2	3	1	2	...	2	5
1	5	1	3	3	3	5	5	2	5	2	...	4	4
2	4	4	3	4	2	5	2	2	3	1	...	1	1
3	6	6	3	1	3	5	6	5	5	1	...	4	2
4	2	6	1	6	1	5	2	5	2	3	...	4	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1837074	2	4	6	5	1	5	6	2	1	3	...	4	1
1837075	6	4	1	1	6	5	2	2	3	4	...	4	3
1837076	1	6	5	2	1	5	1	2	3	4	...	6	1
1837077	4	6	2	2	6	5	2	1	3	3	...	3	6
1837078	1	5	6	4	2	5	2	3	1	5	...	3	6

1837079 rows × 30 columns

In [181…

```
somme = []
for i in range(len(dfX2['pos0'])):
    s = dfX2['f'][i] + dfX2['b'][i] + dfX2['l'][i] + dfX2['r'][i] + dfX2['u']
    somme.append(s)
dfX2['somme'] = somme
dfX2
```

Out[181…

	pos0	pos1	pos2	pos3	pos4	pos5	pos6	pos7	pos8	pos9	...	pos21	pos22
0	3	3	5	1	3	5	2	3	1	2	...	5	4
1	5	1	3	3	3	5	5	2	5	2	...	4	1
2	4	4	3	4	2	5	2	2	3	1	...	1	6
3	6	6	3	1	3	5	6	5	5	1	...	2	4
4	2	6	1	6	1	5	2	5	2	3	...	3	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1837074	2	4	6	5	1	5	6	2	1	3	...	1	3
1837075	6	4	1	1	6	5	2	2	3	4	...	3	5
1837076	1	6	5	2	1	5	1	2	3	4	...	1	3
1837077	4	6	2	2	6	5	2	1	3	3	...	6	4
1837078	1	5	6	4	2	5	2	3	1	5	...	6	1

1837079 rows × 31 columns

In [128…

```
#DecisionTree
from sklearn.tree import DecisionTreeClassifier
from math import sqrt
from sklearn.metrics import precision_score
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
X2_train, X2_test, y_train, y_test = train_test_split(X2, y, test_size=0.10)
dtree = DecisionTreeClassifier(criterion="gini", max_depth=3)
dtree.fit(X2_train, y_train)
ydt = dtree.predict(X2_test)
mean_absolute_error(y_test,ydt)
```

Out[128…

```
0.8587323360985912
```

In [28]:

```
#RandomForest
from sklearn.ensemble import RandomForestClassifier
X2_train, X2_test, y_train, y_test = train_test_split(X2, y, test_size=0.10)
dtree = RandomForestClassifier(max_depth=10,verbose=3)
dtree.fit(X2_train, y_train)
ydt = dtree.predict(X2_test)
mean_absolute_error(y_test,ydt)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
building tree 1 of 100
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.3s remaining:    0.0
s
building tree 2 of 100
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    2.7s remaining:    0.0
```

```
s
building tree 3 of 100
building tree 4 of 100
building tree 5 of 100
building tree 6 of 100
building tree 7 of 100
building tree 8 of 100
building tree 9 of 100
building tree 10 of 100
building tree 11 of 100
building tree 12 of 100
building tree 13 of 100
building tree 14 of 100
building tree 15 of 100
building tree 16 of 100
building tree 17 of 100
building tree 18 of 100
building tree 19 of 100
building tree 20 of 100
building tree 21 of 100
building tree 22 of 100
building tree 23 of 100
building tree 24 of 100
building tree 25 of 100
building tree 26 of 100
building tree 27 of 100
building tree 28 of 100
building tree 29 of 100
building tree 30 of 100
building tree 31 of 100
building tree 32 of 100
building tree 33 of 100
building tree 34 of 100
building tree 35 of 100
building tree 36 of 100
building tree 37 of 100
building tree 38 of 100
building tree 39 of 100
building tree 40 of 100
building tree 41 of 100
building tree 42 of 100
building tree 43 of 100
building tree 44 of 100
building tree 45 of 100
building tree 46 of 100
building tree 47 of 100
building tree 48 of 100
building tree 49 of 100
building tree 50 of 100
building tree 51 of 100
building tree 52 of 100
building tree 53 of 100
building tree 54 of 100
building tree 55 of 100
building tree 56 of 100
building tree 57 of 100
building tree 58 of 100
building tree 59 of 100
building tree 60 of 100
building tree 61 of 100
building tree 62 of 100
building tree 63 of 100
building tree 64 of 100
building tree 65 of 100
```

```
building tree 66 of 100
building tree 67 of 100
building tree 68 of 100
building tree 69 of 100
building tree 70 of 100
building tree 71 of 100
building tree 72 of 100
building tree 73 of 100
building tree 74 of 100
building tree 75 of 100
building tree 76 of 100
building tree 77 of 100
building tree 78 of 100
building tree 79 of 100
building tree 80 of 100
building tree 81 of 100
building tree 82 of 100
building tree 83 of 100
building tree 84 of 100
building tree 85 of 100
building tree 86 of 100
building tree 87 of 100
building tree 88 of 100
building tree 89 of 100
building tree 90 of 100
building tree 91 of 100
building tree 92 of 100
building tree 93 of 100
building tree 94 of 100
building tree 95 of 100
building tree 96 of 100
building tree 97 of 100
building tree 98 of 100
building tree 99 of 100
building tree 100 of 100
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  2.2min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0
s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0
s
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.8s finished
```

Out[28]:
```
0.8584873821499336
```

Regroupement des colonnes

In [182…

```
z = []
for i in range(np.size(y)):
    if y[i] == 12 or y[i] == 13 or y[i] == 14:
        z.append(0)
    elif y[i] == 10 or y[i] == 9 or y[i] == 8:
        z.append(1)
    else:
        z.append(2)
z = np.array(z)
np.bincount(z)
```

Out[182…
```
array([436546, 702622, 697911])
```

In [199…

```
X2 = dfX2.to_numpy()
#RandomForest
from sklearn.ensemble import RandomForestClassifier
X2_train, X2_test, z_train, z_test = train_test_split(X2, z, test_size=0.30)
dtree = RandomForestClassifier(max_depth=10,verbose=3, n_jobs=50)
dtree.fit(X2_train, z_train)
zdt = dtree.predict(X2_test)
mean_absolute_error(z_test,zdt)
```

```
[Parallel(n_jobs=50)]: Using backend ThreadingBackend with 50 concurrent worke
rs.
building tree 1 of 100
building tree 2 of 100
building tree 3 of 100building tree 4 of 100building tree 5 of 100
building tree 6 of 100


building tree 7 of 100
building tree 8 of 100
building tree 9 of 100
building tree 10 of 100
building tree 11 of 100building tree 12 of 100building tree 13 of 100
building tree 14 of 100

building tree 15 of 100
building tree 16 of 100

building tree 17 of 100
building tree 18 of 100
building tree 19 of 100building tree 20 of 100
building tree 21 of 100

building tree 22 of 100building tree 23 of 100

building tree 24 of 100
building tree 25 of 100
building tree 26 of 100building tree 27 of 100building tree 28 of 100building
tree 29 of 100

building tree 30 of 100building tree 31 of 100



building tree 32 of 100building tree 33 of 100

building tree 34 of 100building tree 35 of 100

building tree 36 of 100building tree 37 of 100
building tree 38 of 100

building tree 39 of 100building tree 40 of 100
building tree 41 of 100building tree 42 of 100


building tree 43 of 100building tree 44 of 100building tree 45 of 100
building tree 47 of 100
building tree 46 of 100building tree 48 of 100


building tree 49 of 100
building tree 50 of 100

building tree 51 of 100
building tree 52 of 100
```

```
building tree 53 of 100
building tree 54 of 100
building tree 55 of 100
building tree 56 of 100
building tree 57 of 100
building tree 58 of 100
building tree 59 of 100building tree 60 of 100

building tree 61 of 100
building tree 62 of 100
building tree 63 of 100
building tree 64 of 100
building tree 65 of 100
building tree 66 of 100
building tree 67 of 100building tree 68 of 100building tree 69 of 100
building tree 70 of 100

building tree 71 of 100building tree 72 of 100building tree 73 of 100
building tree 74 of 100


building tree 75 of 100building tree 76 of 100building tree 77 of 100



building tree 78 of 100
building tree 79 of 100
building tree 80 of 100building tree 81 of 100

building tree 82 of 100
building tree 83 of 100
building tree 84 of 100building tree 85 of 100

building tree 86 of 100
building tree 87 of 100
building tree 88 of 100
building tree 89 of 100
[Parallel(n_jobs=50)]: Done  35 out of 100 | elapsed:   13.3s remaining:   24.
6s
building tree 90 of 100
building tree 91 of 100
building tree 92 of 100
building tree 93 of 100
building tree 94 of 100
building tree 95 of 100
building tree 96 of 100
building tree 97 of 100
building tree 98 of 100
building tree 99 of 100
building tree 100 of 100
[Parallel(n_jobs=50)]: Done  69 out of 100 | elapsed:   22.9s remaining:   10.
3s
[Parallel(n_jobs=50)]: Done 100 out of 100 | elapsed:   23.8s finished
[Parallel(n_jobs=50)]: Using backend ThreadingBackend with 50 concurrent worke
rs.
[Parallel(n_jobs=50)]: Done  35 out of 100 | elapsed:    0.5s remaining:    0.
9s
[Parallel(n_jobs=50)]: Done  69 out of 100 | elapsed:    0.7s remaining:    0.
3s
[Parallel(n_jobs=50)]: Done 100 out of 100 | elapsed:    0.8s finished
```

Out[199…

```
0.6641028153373832
```

In [196…

```
#DecisionTree
X2_train, X2_test, z_train, z_test = train_test_split(X2, z, test_size=0.30)
dtree = DecisionTreeClassifier(criterion="gini", max_depth=5)
dtree.fit(X2_train, z_train)
zdt = dtree.predict(X2_test)
mean_absolute_error(z_test,zdt)
```

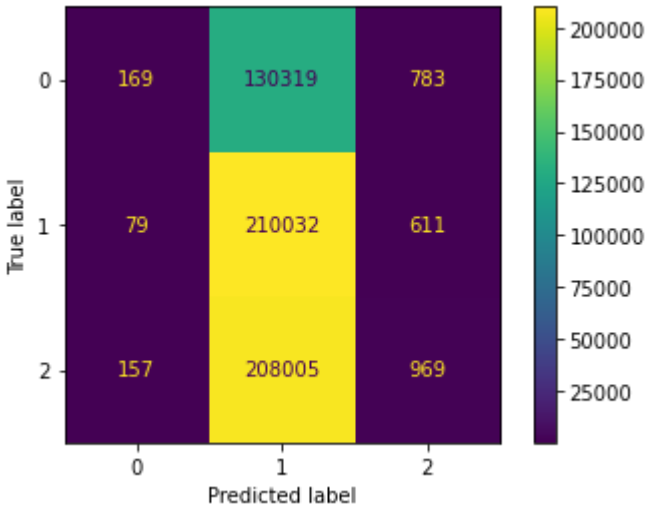
Out[196…
```
0.6185431953607536
```

In [200…
```
np.bincount(zdt)
```

Out[200…
```
array([   607, 441847, 108670])
```

In [198…
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(dtree,X2_test,z_test)
```

Out[198…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f9b419774
00>
```



In [51]:
```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(hidden_layer_sizes=50,random_state=1, max_iter=20,solver=
clf.fit(X_train, z_train)
```

```
Iteration 1, loss = 0.69631506
Iteration 2, loss = 0.69392003
Iteration 3, loss = 0.69373080
Iteration 4, loss = 0.69365517
Iteration 5, loss = 0.69357592
Iteration 6, loss = 0.69353350
Iteration 7, loss = 0.69345259
Iteration 8, loss = 0.69339986
Iteration 9, loss = 0.69336452
Iteration 10, loss = 0.69331956
Iteration 11, loss = 0.69331658
Iteration 12, loss = 0.69328718
Iteration 13, loss = 0.69325883
Iteration 14, loss = 0.69325440
Training loss did not improve more than tol=0.000100 for 10 consecutive epoch
s. Stopping.
```

Out[51]:
```
MLPClassifier(hidden_layer_sizes=50, max_iter=20, random_state=1, solver='sg
d',
              verbose=10)
```

In [52]:

```
zdt = clf.predict(X2_test)
mean_absolute_error(z_test,zdt)
```

Out[52]:
```
0.5006368802665099
```

In [53]:
```
np.bincount(zdt)
```

Out[53]:
```
array([ 51208, 132500])
```

Modélisation des données sous la forme de coins

Les 6 cellules qui suivent sont un simple tatonnement pour pouvoir faire la fonction cornerCube() expliquée plus tard.

In [283…
```
print(X2[365030],X2[411427],X2[1266419])
np.split(X2[411427],6)
```

```
[3 3 1 1 5 5 6 6 1 1 5 5 6 6 3 3 4 4 4 4 2 2 2 2] [3 4 3 4 2 5 2 5 1 1 1 1 6 6
6 6 4 5 4 5 2 3 2 3] [3 3 3 3 5 5 5 5 2 1 2 1 4 6 4 6 4 4 1 1 6 6 2 2]
```

Out[283…
```
[array([3, 4, 3, 4]),
 array([2, 5, 2, 5]),
 array([1, 1, 1, 1]),
 array([6, 6, 6, 6]),
 array([4, 5, 4, 5]),
 array([2, 3, 2, 3])]
```

In [ ]:
```
#[(5'5',6'13',4'16'),(1'9',2'4',5'17'),(6'12',3'0',4'18'),(4'1',1'8',5'19'),(
```

In [284…
```
np.split(X2[365030],6)
```

Out[284…
```
[array([3, 3, 1, 1]),
 array([5, 5, 6, 6]),
 array([1, 1, 5, 5]),
 array([6, 6, 3, 3]),
 array([4, 4, 4, 4]),
 array([2, 2, 2, 2])]
```

In [ ]:
```
#[(5'5',6'13',4'16'),(1'9',5'4',4'17'),(6'12',3'0',4'18'),(3'1',1'8',4'19'),(
```

In [314…
```
np.split(X2[1266419],6)
```

Out[314…
```
[array([3, 3, 3, 3]),
 array([5, 5, 5, 5]),
 array([2, 1, 2, 1]),
 array([4, 6, 4, 6]),
 array([4, 4, 1, 1]),
 array([6, 6, 2, 2])]
```

In [ ]:
```
#[(5'5',6'13',4'16'),(1'9',5'4',4'17'),(4'12',3'0',1'18'),(3'1',2'8',1'19'),(
```

In [52]:
```
def cornerCube(dfX):
    df = pd.DataFrame()
    df1 = faceCube(dfX)
```

```
    df['corner0'] = df1[['pos5','pos13','pos16']].apply(tuple, axis=1)
    df['corner1'] = df1[['pos9','pos4','pos17']].apply(tuple, axis=1)
    df['corner2'] = df1[['pos1','pos8','pos19']].apply(tuple, axis=1)
    df['corner3'] = df1[['pos12','pos0','pos18']].apply(tuple, axis=1)
    df['corner4'] = df1[['pos10','pos3','pos21']].apply(tuple, axis=1)
    df['corner5'] = df1[['pos2','pos14','pos20']].apply(tuple, axis=1)
    df['corner6'] = df1[['pos15','pos7','pos22']].apply(tuple, axis=1)
    df['corner7'] = df1[['pos6','pos11','pos23']].apply(tuple, axis=1)
    return df
```

In [5]:

```
dfX3 = cornerCube(dfX)
dfX3
```

Out[5]:

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7
0	(5, 6, 4)	(2, 3, 6)	(3, 1, 4)	(1, 3, 2)	(4, 1, 5)	(5, 1, 2)	(6, 3, 4)	(2, 6, 5)
1	(5, 6, 4)	(2, 3, 6)	(1, 5, 4)	(2, 5, 1)	(6, 3, 4)	(3, 1, 4)	(3, 2, 1)	(5, 2, 6)
2	(5, 6, 4)	(1, 2, 5)	(4, 3, 1)	(3, 4, 6)	(5, 4, 1)	(3, 2, 1)	(5, 2, 6)	(2, 3, 6)
3	(5, 6, 4)	(1, 3, 2)	(6, 5, 2)	(4, 6, 3)	(5, 1, 2)	(3, 1, 4)	(1, 5, 4)	(6, 2, 3)
4	(5, 6, 4)	(3, 1, 4)	(6, 2, 3)	(1, 2, 5)	(4, 6, 3)	(1, 5, 4)	(6, 5, 2)	(2, 1, 3)
...	...	...	...	...	...	...	...	...
1837074	(5, 6, 4)	(3, 1, 4)	(4, 1, 5)	(3, 2, 1)	(2, 5, 1)	(6, 3, 4)	(6, 2, 3)	(6, 5, 2)
1837075	(5, 6, 4)	(4, 6, 3)	(4, 3, 1)	(3, 6, 2)	(2, 1, 3)	(1, 5, 4)	(1, 2, 5)	(2, 6, 5)
1837076	(5, 6, 4)	(4, 1, 5)	(6, 3, 4)	(3, 1, 4)	(3, 2, 1)	(5, 2, 6)	(6, 2, 3)	(1, 2, 5)
1837077	(5, 6, 4)	(3, 6, 2)	(6, 3, 4)	(5, 4, 1)	(5, 2, 6)	(2, 1, 3)	(3, 1, 4)	(2, 5, 1)
1837078	(5, 6, 4)	(5, 2, 6)	(5, 1, 2)	(4, 1, 5)	(3, 4, 6)	(6, 2, 3)	(4, 3, 1)	(2, 1, 3)

1837079 rows × 8 columns

In [54]:

```
def myNumerize(L):
    if L == [4,5,6]:
        return 0
    elif L == [1,2,3]:
        return 1
    elif L == [1,2,5]:
        return 2
    elif L == [1,3,4]:
        return 3
    elif L == [1,4,5]:
        return 4
    elif L == [2,3,6]:
        return 5
    elif L == [2,5,6]:
        return 6
    if L == [3,4,6]:
        return 7
```

In [7]:

```
for col in dfX3.columns:
    dfX3[col] = dfX3[col].apply(sorted)
    dfX3[col] = dfX3[col].apply(myNumerize)
dfX3
```

Out[7]:

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7
0	0	5	3	1	4	2	7	6
1	0	5	4	2	7	3	1	6
2	0	2	3	7	4	1	6	5
3	0	1	6	7	2	3	4	5
4	0	3	5	2	7	4	6	1
...	...	...	...	...	...	...	...	...
1837074	0	3	4	1	2	7	5	6
1837075	0	7	3	5	1	4	2	6
1837076	0	4	7	3	1	6	5	2
1837077	0	5	7	4	6	1	3	2
1837078	0	6	2	4	7	5	3	1

1837079 rows × 8 columns

In [77]:

```
#RandomForest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
X_train, X_test, y_train, y_test = train_test_split(dfX3.to_numpy(), y, test_
dtree = RandomForestClassifier(max_depth=80,verbose=3,n_jobs=32)
dtree.fit(X_train, y_train)
ydr = dtree.predict(X_train)
ydt = dtree.predict(X_test)
print(mean_absolute_error(y_train,ydr),mean_absolute_error(y_test,ydt))
```

```
[Parallel(n_jobs=32)]: Using backend ThreadingBackend with 32 concurrent worke
rs.
building tree 1 of 100
building tree 2 of 100
building tree 3 of 100
building tree 4 of 100building tree 5 of 100

building tree 6 of 100building tree 7 of 100
building tree 8 of 100

building tree 9 of 100building tree 10 of 100

building tree 11 of 100building tree 12 of 100
building tree 13 of 100
building tree 14 of 100

building tree 15 of 100building tree 16 of 100
building tree 17 of 100building tree 18 of 100


building tree 19 of 100
building tree 20 of 100
building tree 21 of 100
building tree 22 of 100
building tree 23 of 100
building tree 24 of 100building tree 25 of 100building tree 26 of 100
```

```
building tree 27 of 100building tree 28 of 100


building tree 29 of 100building tree 30 of 100

building tree 31 of 100
building tree 32 of 100

building tree 33 of 100building tree 34 of 100

building tree 35 of 100
building tree 36 of 100
building tree 37 of 100
building tree 38 of 100
building tree 39 of 100building tree 40 of 100

building tree 41 of 100
building tree 42 of 100
building tree 43 of 100
building tree 44 of 100
building tree 45 of 100
building tree 46 of 100
building tree 47 of 100
building tree 48 of 100
building tree 49 of 100
building tree 50 of 100building tree 51 of 100

building tree 52 of 100
building tree 53 of 100building tree 54 of 100
building tree 55 of 100building tree 56 of 100

building tree 57 of 100

building tree 58 of 100
building tree 59 of 100
building tree 60 of 100
building tree 61 of 100
building tree 62 of 100building tree 63 of 100

building tree 64 of 100
building tree 65 of 100
building tree 66 of 100
building tree 68 of 100
building tree 67 of 100building tree 69 of 100

building tree 70 of 100
building tree 71 of 100
building tree 72 of 100building tree 73 of 100

building tree 74 of 100
building tree 75 of 100
building tree 76 of 100
building tree 77 of 100
building tree 78 of 100
building tree 79 of 100
building tree 80 of 100
building tree 81 of 100
building tree 82 of 100
building tree 83 of 100
building tree 84 of 100
building tree 85 of 100
building tree 86 of 100
building tree 87 of 100
building tree 88 of 100building tree 89 of 100
```

```
building tree 90 of 100

building tree 91 of 100
building tree 92 of 100
building tree 93 of 100building tree 94 of 100

building tree 95 of 100
building tree 96 of 100
building tree 97 of 100
building tree 98 of 100
building tree 99 of 100building tree 100 of 100

[Parallel(n_jobs=32)]: Done  71 out of 100 | elapsed:   33.5s remaining:   13.
7s
[Parallel(n_jobs=32)]: Done 100 out of 100 | elapsed:   35.5s finished
[Parallel(n_jobs=32)]: Using backend ThreadingBackend with 32 concurrent worke
rs.
[Parallel(n_jobs=32)]: Done  71 out of 100 | elapsed:   20.4s remaining:    8.
3s
[Parallel(n_jobs=32)]: Done 100 out of 100 | elapsed:   22.0s finished
[Parallel(n_jobs=32)]: Using backend ThreadingBackend with 32 concurrent worke
rs.
0.7807164877090502 0.7894266988917195
[Parallel(n_jobs=32)]: Done  71 out of 100 | elapsed:    0.6s remaining:    0.
2s
[Parallel(n_jobs=32)]: Done 100 out of 100 | elapsed:    0.7s finished
```

In [13]:

```
dfX4 = dfX3.copy()
dfX4['distance'] = dfY
dfX4 = dfX4.drop_duplicates()
dfY2 = dfX4['distance']
```

In [14]:

```
dfX44 = dfX3.copy()
dfX44['distance'] = dfY
X4 = dfX4.to_numpy()
X4 = X4.tolist()
X44 = dfX44.to_numpy()
X44 = X44.tolist()
weight = []
for i in range(len(X4)):
    weight.append(X44.count(X4[i]))
    if i%1000 == 0:
        print(i)
```

```
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
```

```
17000
18000
19000
20000
```

In [30]:
```
dfX4['weight'] = np.array(weight)
dfX5 = dfX4.sort_values(by=['weight'])
```

In [31]:
```
dfX5 = dfX5.drop_duplicates(subset=['corner0','corner1','corner2','corner3','
y2 = dfX5['distance'].values
dfX5 = dfX5.drop('distance',axis=1)
dfX5 = dfX5.drop('weight',axis=1)
dfX5
```

Out[31]:

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7
6472	0	6	2	4	1	3	7	5
16613	0	5	4	7	6	1	3	2
12583	0	7	5	6	1	2	4	3
6774	0	1	6	7	4	3	2	5
24108	0	7	5	1	4	6	3	2
...	...	...	...	...	...	...	...	...
1756	0	3	5	6	7	2	1	4
3467	0	6	7	5	1	3	4	2
140	0	3	7	1	4	2	5	6
150	0	5	7	4	1	2	3	6
6617	0	6	2	5	3	4	7	1

5040 rows × 8 columns

In [131…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
X_train, X_test, y_train, y_test = train_test_split(dfX5.to_numpy(), y2, test_
dtree = RandomForestClassifier(max_depth=5)
dtree.fit(X_train, y_train)
ydr = dtree.predict(X_train)
ydt = dtree.predict(X_test)
print(mean_absolute_error(y_train,ydr),mean_absolute_error(y_test,ydt))
```

```
0.4928571428571429 0.5087301587301587
```

In [132…
```
from sklearn.neural_network import MLPClassifier
X_train, X_test, y_train, y_test = train_test_split(dfX5.to_numpy(), y2, test_
mlp = MLPClassifier(hidden_layer_sizes=800,random_state=1, max_iter=500,solve
mlp.fit(X_train, y_train)
ydr = mlp.predict(X_train)
ydt = mlp.predict(X_test)
print(mean_absolute_error(y_train,ydr),mean_absolute_error(y_test,ydt))
```

```
0.49603174603174605 0.5154761904761904
```

In [85]:
```
X_train = X_train.tolist()
X_test = X_test.tolist()
for i in range(len(X_test)):
    if X_test[i] in X_train:
        j = X_train.index(X_test[i])
        ydt[i] = y_train[j]
mean_absolute_error(y_test,ydt)
```

Out[85]:
```
0.5015873015873016
```

Ajout des orientations de couleur (n'es pas implémenté sur les données finales)

In [44]:
```
dfX6 = coinCube(dfX)
```

In [47]:
```
def compressTuple(L):
    if L == (5,6,4):
        return (0,0)
    elif L == (1,3,2):
        return (1,0)
    elif L == (3,2,1):
        return (1,1)
    elif L == (2,1,3):
        return (1,2)
    elif L == (1,2,5):
        return (2,0)
    elif L == (2,5,1):
        return (2,1)
    elif L == (5,1,2):
        return (2,2)
    elif L == (1,4,3):
        return (3,0)
    elif L == (4,3,1):
        return (3,1)
    elif L == (3,1,4):
        return (3,2)
    elif L == (1,5,4):
        return (4,0)
    elif L == (5,4,1):
        return (4,1)
    elif L == (4,1,5):
        return (4,2)
    elif L == (2,3,6):
        return (5,0)
    elif L == (3,6,2):
        return (5,1)
    elif L == (6,2,3):
        return (5,2)
    elif L == (2,6,5):
        return (6,0)
    elif L == (6,5,2):
        return (6,1)
    elif L == (5,2,6):
        return (6,2)
    elif L == (3,4,6):
        return (7,0)
    elif L == (4,6,3):
        return (7,1)
```

```
    elif L == (6,3,4):
        return (7,2)
    else:
        L == None
```

In [48]:

```
for col in dfX6.columns:
    dfX6[col] = dfX6[col].apply(compressTuple)
dfX6
```

Out[48]:

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7
0	(0, 0)	(5, 0)	(3, 2)	(1, 0)	(4, 2)	(2, 2)	(7, 2)	(6, 0)
1	(0, 0)	(5, 0)	(4, 0)	(2, 1)	(7, 2)	(3, 2)	(1, 1)	(6, 2)
2	(0, 0)	(2, 0)	(3, 1)	(7, 0)	(4, 1)	(1, 1)	(6, 2)	(5, 0)
3	(0, 0)	(1, 0)	(6, 1)	(7, 1)	(2, 2)	(3, 2)	(4, 0)	(5, 2)
4	(0, 0)	(3, 2)	(5, 2)	(2, 0)	(7, 1)	(4, 0)	(6, 1)	(1, 2)
...	...	...	...	...	...	...	...	...
1837074	(0, 0)	(3, 2)	(4, 2)	(1, 1)	(2, 1)	(7, 2)	(5, 2)	(6, 1)
1837075	(0, 0)	(7, 1)	(3, 1)	(5, 1)	(1, 2)	(4, 0)	(2, 0)	(6, 0)
1837076	(0, 0)	(4, 2)	(7, 2)	(3, 2)	(1, 1)	(6, 2)	(5, 2)	(2, 0)
1837077	(0, 0)	(5, 1)	(7, 2)	(4, 1)	(6, 2)	(1, 2)	(3, 2)	(2, 1)
1837078	(0, 0)	(6, 2)	(2, 2)	(4, 2)	(7, 0)	(5, 2)	(3, 1)	(1, 2)

1837079 rows × 8 columns

In [51]:

```
i=0
for col in dfX6.columns:
    nom = 'orient' + str(i)
    dfX6[[col,nom]] = pd.DataFrame(dfX6[col].tolist(),index=dfX6.index)
    i+=1
dfX6
```

Out[51]:

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7	orient0	o
0	0	5	3	1	4	2	7	6	0	
1	0	5	4	2	7	3	1	6	0	
2	0	2	3	7	4	1	6	5	0	
3	0	1	6	7	2	3	4	5	0	
4	0	3	5	2	7	4	6	1	0	
...	...	...	...	...	...	...	...	...	...	...
1837074	0	3	4	1	2	7	5	6	0	
1837075	0	7	3	5	1	4	2	6	0	
1837076	0	4	7	3	1	6	5	2	0	
1837077	0	5	7	4	6	1	3	2	0	

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7	orient0	o
1837078	0	6	2	4	7	5	3	1	0	

1837079 rows × 16 columns

In [60]:
```
dfX6 = dfX6.drop('corner0',axis=1)
dfX6 = dfX6.drop('orient0',axis=1)
```

In [61]:
```
#RandomForest
from sklearn.ensemble import RandomForestClassifier
X_train, X_test, y_train, y_test = train_test_split(dfX6.to_numpy(), y, test_
dtree = RandomForestClassifier(max_depth = 20 ,verbose=3, n_jobs=50)
dtree.fit(X_train, y_train)
ydtr = dtree.predict(X_train)
ydt = dtree.predict(X_test)
print(mean_absolute_error(y_train,ydtr),mean_absolute_error(y_test,ydt))
```

```
[Parallel(n_jobs=50)]: Using backend ThreadingBackend with 50 concurrent worke
rs.
building tree 1 of 100building tree 2 of 100

building tree 3 of 100
building tree 4 of 100
building tree 5 of 100
building tree 6 of 100
building tree 7 of 100building tree 8 of 100
building tree 9 of 100building tree 10 of 100
building tree 11 of 100

building tree 12 of 100

building tree 13 of 100building tree 14 of 100

building tree 15 of 100building tree 16 of 100
building tree 17 of 100
building tree 18 of 100

building tree 19 of 100
building tree 20 of 100
building tree 21 of 100building tree 22 of 100

building tree 23 of 100building tree 24 of 100

building tree 25 of 100
building tree 26 of 100building tree 27 of 100

building tree 28 of 100
building tree 29 of 100
building tree 30 of 100building tree 31 of 100
building tree 32 of 100

building tree 33 of 100
building tree 34 of 100
building tree 35 of 100
building tree 36 of 100building tree 37 of 100
building tree 38 of 100
building tree 39 of 100
building tree 40 of 100building tree 41 of 100
```

```
building tree 42 of 100building tree 43 of 100
building tree 44 of 100

building tree 45 of 100
building tree 46 of 100
building tree 47 of 100
building tree 48 of 100
building tree 49 of 100

building tree 50 of 100
building tree 51 of 100
building tree 52 of 100building tree 53 of 100

building tree 54 of 100building tree 55 of 100building tree 56 of 100
building tree 57 of 100

building tree 58 of 100building tree 59 of 100
building tree 60 of 100


building tree 61 of 100building tree 62 of 100
building tree 63 of 100

building tree 64 of 100
building tree 65 of 100building tree 66 of 100building tree 67 of 100


building tree 68 of 100building tree 69 of 100

building tree 70 of 100
building tree 71 of 100
building tree 72 of 100
building tree 73 of 100building tree 74 of 100building tree 75 of 100

building tree 76 of 100

building tree 77 of 100
building tree 78 of 100building tree 79 of 100building tree 80 of 100
building tree 81 of 100

building tree 82 of 100building tree 83 of 100building tree 84 of 100

building tree 85 of 100building tree 86 of 100building tree 87 of 100building
tree 88 of 100

building tree 89 of 100
building tree 90 of 100



building tree 91 of 100

building tree 92 of 100
[Parallel(n_jobs=50)]: Done  35 out of 100 | elapsed:   14.2s remaining:   26.
4s
building tree 93 of 100
building tree 94 of 100
building tree 95 of 100building tree 96 of 100
building tree 97 of 100
building tree 98 of 100building tree 99 of 100


building tree 100 of 100
```

```
[Parallel(n_jobs=50)]: Done  69 out of 100 | elapsed:   27.4s remaining:   12.
3s
[Parallel(n_jobs=50)]: Done 100 out of 100 | elapsed:   28.0s finished
[Parallel(n_jobs=50)]: Using backend ThreadingBackend with 50 concurrent worke
rs.
[Parallel(n_jobs=50)]: Done  35 out of 100 | elapsed:   25.5s remaining:   47.
4s
[Parallel(n_jobs=50)]: Done  69 out of 100 | elapsed:   30.5s remaining:   13.
7s
[Parallel(n_jobs=50)]: Done 100 out of 100 | elapsed:   32.1s finished
[Parallel(n_jobs=50)]: Using backend ThreadingBackend with 50 concurrent worke
rs.
[Parallel(n_jobs=50)]: Done  35 out of 100 | elapsed:   28.0s remaining:   52.
0s
[Parallel(n_jobs=50)]: Done  69 out of 100 | elapsed:   32.7s remaining:   14.
7s
0.07402734124517305 0.9035240708080214
[Parallel(n_jobs=50)]: Done 100 out of 100 | elapsed:   34.4s finished
```

Premier résultat envoyé : colonne constante de 11

In [ ]:

```
dfY2 = dfY.copy()
dfY2['distance'] = 11
dfY2.loc[3674158] = 11
dfY2 = pd.concat([dfY,dfY2], ignore_index = True)
dfY2 = dfY2.iloc[1837079:]
dfY2.to_csv("y_test.csv")
dfY2
```

Deuxième résultat envoyé : utilisation des coins

In [142…

```
dfT3 = cornerCube(dfT)
```

In [143…

```
for col in dfT3.columns:
    dfT3[col] = dfT3[col].apply(sorted)
    dfT3[col] = dfT3[col].apply(myNumerize)
dfT3
```

Out[143…

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7
1837079	0	7	3	6	5	2	4	1
1837080	0	6	3	1	5	4	7	2
1837081	0	1	7	6	2	3	4	5
1837082	0	2	7	5	3	1	6	4
1837083	0	1	4	2	6	7	3	5
…	…	…	…	…	…	…	…	…
3674154	0	5	4	1	6	2	7	3

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7
3674155	0	6	2	1	5	4	7	3
3674156	0	7	6	5	3	4	1	2
3674157	0	4	5	7	2	1	3	6
3674158	0	5	4	2	3	1	7	6

1837080 rows × 8 columns

In [144…
```
T3 = dfT3.to_numpy()
```

In [468…
```
dtree = RandomForestClassifier(max_depth=100,verbose=3, n_jobs=32)
dtree.fit(X3, y)
ydt = dtree.predict(T3)
```

```
[Parallel(n_jobs=32)]: Using backend ThreadingBackend with 32 concurrent worke
rs.
building tree 1 of 100building tree 2 of 100

building tree 3 of 100
building tree 4 of 100
building tree 5 of 100
building tree 6 of 100building tree 7 of 100building tree 8 of 100


building tree 9 of 100building tree 10 of 100building tree 11 of 100

building tree 12 of 100

building tree 13 of 100building tree 14 of 100
building tree 15 of 100
building tree 16 of 100
building tree 17 of 100
building tree 18 of 100building tree 19 of 100building tree 20 of 100


building tree 21 of 100

building tree 22 of 100building tree 23 of 100building tree 24 of 100


building tree 25 of 100building tree 26 of 100

building tree 28 of 100building tree 27 of 100
building tree 29 of 100

building tree 30 of 100
building tree 31 of 100building tree 32 of 100

building tree 33 of 100
building tree 34 of 100
building tree 35 of 100
building tree 36 of 100
building tree 37 of 100
building tree 38 of 100
building tree 39 of 100
building tree 40 of 100
```

```
building tree 41 of 100
building tree 42 of 100
building tree 43 of 100
building tree 44 of 100
building tree 45 of 100
building tree 46 of 100
building tree 47 of 100
building tree 48 of 100
building tree 49 of 100
building tree 50 of 100
building tree 51 of 100
building tree 52 of 100
building tree 53 of 100
building tree 54 of 100
building tree 55 of 100
building tree 56 of 100building tree 57 of 100

building tree 58 of 100building tree 59 of 100

building tree 60 of 100
building tree 61 of 100
building tree 62 of 100building tree 63 of 100

building tree 64 of 100
building tree 65 of 100
building tree 66 of 100
building tree 67 of 100
building tree 68 of 100
building tree 69 of 100
building tree 70 of 100
building tree 71 of 100
building tree 72 of 100
building tree 73 of 100building tree 74 of 100
building tree 75 of 100

building tree 76 of 100
building tree 77 of 100
building tree 78 of 100
building tree 79 of 100
building tree 80 of 100
building tree 81 of 100building tree 82 of 100

building tree 83 of 100
building tree 84 of 100
building tree 85 of 100building tree 86 of 100

building tree 87 of 100
building tree 88 of 100
building tree 89 of 100
building tree 90 of 100
building tree 91 of 100
building tree 92 of 100
building tree 93 of 100
building tree 94 of 100building tree 95 of 100

building tree 96 of 100
building tree 97 of 100
building tree 98 of 100
building tree 99 of 100
building tree 100 of 100
[Parallel(n_jobs=32)]: Done  71 out of 100 | elapsed:   48.7s remaining:   19.
9s
[Parallel(n_jobs=32)]: Done 100 out of 100 | elapsed:   51.3s finished
[Parallel(n_jobs=32)]: Using backend ThreadingBackend with 32 concurrent worke
```

```
rs.
[Parallel(n_jobs=32)]: Done  71 out of 100 | elapsed:   24.8s remaining:   10.
1s
[Parallel(n_jobs=32)]: Done 100 out of 100 | elapsed:   26.6s finished
```

In [470…

```
np.bincount(ydt)
```

Out[470…

```
array([     0,      0,      0,      0,      0,      0,      0,      0,
            0,      0, 665173, 918540, 253367])
```

In [154…

```
dfYt = pd.DataFrame(ydt)
ids = pd.DataFrame(np.arange(1837079,3674159))
ids['ID'] = ids
dfYt.index = ids['ID']
dfYt
```

Out[154…

ID	0
1837079	10
1837080	12
1837081	12
1837082	10
1837083	11
…	…
3674154	11
3674155	11
3674156	10
3674157	12
3674158	11

1837080 rows × 1 columns

In [153…

```
dfYt.to_csv("y_test.csv")
```

In [145…

```
X5 = dfX5.to_numpy()
dfT3
```

Out[145…

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7
1837079	0	7	3	6	5	2	4	1
1837080	0	6	3	1	5	4	7	2
1837081	0	1	7	6	2	3	4	5
1837082	0	2	7	5	3	1	6	4
1837083	0	1	4	2	6	7	3	5
…	…	…	…	…	…	…	…	…

ID	corner0	corner1	corner2	corner3	corner4	corner5	corner6	corner7
3674154	0	5	4	1	6	2	7	3
3674155	0	6	2	1	5	4	7	3
3674156	0	7	6	5	3	4	1	2
3674157	0	4	5	7	2	1	3	6
3674158	0	5	4	2	3	1	7	6

1837080 rows × 8 columns

In [147…

```
dtree = RandomForestClassifier(max_depth = 100, verbose = 3)
dtree.fit(X5, y2)
ydt = dtree.predict(T3)
np.bincount(ydt)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0
s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0
s
building tree 1 of 100
building tree 2 of 100
building tree 3 of 100
building tree 4 of 100
building tree 5 of 100
building tree 6 of 100
building tree 7 of 100
building tree 8 of 100
building tree 9 of 100
building tree 10 of 100
building tree 11 of 100
building tree 12 of 100
building tree 13 of 100
building tree 14 of 100
building tree 15 of 100
building tree 16 of 100
building tree 17 of 100
building tree 18 of 100
building tree 19 of 100
building tree 20 of 100
building tree 21 of 100
building tree 22 of 100
building tree 23 of 100
building tree 24 of 100
building tree 25 of 100
building tree 26 of 100
building tree 27 of 100
building tree 28 of 100
building tree 29 of 100
building tree 30 of 100
building tree 31 of 100
building tree 32 of 100
building tree 33 of 100
building tree 34 of 100
building tree 35 of 100
building tree 36 of 100
```

```
building tree 37 of 100
building tree 38 of 100
building tree 39 of 100
building tree 40 of 100
building tree 41 of 100
building tree 42 of 100
building tree 43 of 100
building tree 44 of 100
building tree 45 of 100
building tree 46 of 100
building tree 47 of 100
building tree 48 of 100
building tree 49 of 100
building tree 50 of 100
building tree 51 of 100
building tree 52 of 100
building tree 53 of 100
building tree 54 of 100
building tree 55 of 100
building tree 56 of 100
building tree 57 of 100
building tree 58 of 100
building tree 59 of 100
building tree 60 of 100
building tree 61 of 100
building tree 62 of 100
building tree 63 of 100
building tree 64 of 100
building tree 65 of 100
building tree 66 of 100
building tree 67 of 100
building tree 68 of 100
building tree 69 of 100
building tree 70 of 100
building tree 71 of 100
building tree 72 of 100
building tree 73 of 100
building tree 74 of 100
building tree 75 of 100
building tree 76 of 100
building tree 77 of 100
building tree 78 of 100
building tree 79 of 100
building tree 80 of 100
building tree 81 of 100
building tree 82 of 100
building tree 83 of 100
building tree 84 of 100
building tree 85 of 100
building tree 86 of 100
building tree 87 of 100
building tree 88 of 100
building tree 89 of 100
building tree 90 of 100
building tree 91 of 100
building tree 92 of 100
building tree 93 of 100
building tree 94 of 100
building tree 95 of 100
building tree 96 of 100
building tree 97 of 100
building tree 98 of 100
building tree 99 of 100
building tree 100 of 100
```

```
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0
s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.4s remaining:    0.0
s
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:   22.0s finished
```

Out[147…
```
array([     0,      0,      0,      0,      0,      0,      0,      0,
            0,      0, 663711, 918540, 254829])
```

In [148…
```
#Correction de doublons dans le test
X5l = X5.tolist()
T3l = T3.tolist()
for i in range(len(T3l)):
    if T3l[i] in X5l:
        j = X5l.index(T3l[i])
        ydt[i] = y2[j]
        if i%100000==0:
            print(i)
```

```
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1000000
1100000
1200000
1300000
1400000
1500000
1600000
1700000
1800000
```

In [149…
```
np.bincount(ydt)
```

Out[149…
```
array([     0,      0,      0,      0,      0,      0,      0,      0,
            0,      0, 663711, 918540, 254829])
```