TME 2 - Estimation de densité

CHERCHOUR Lièce & DIEZ Marie

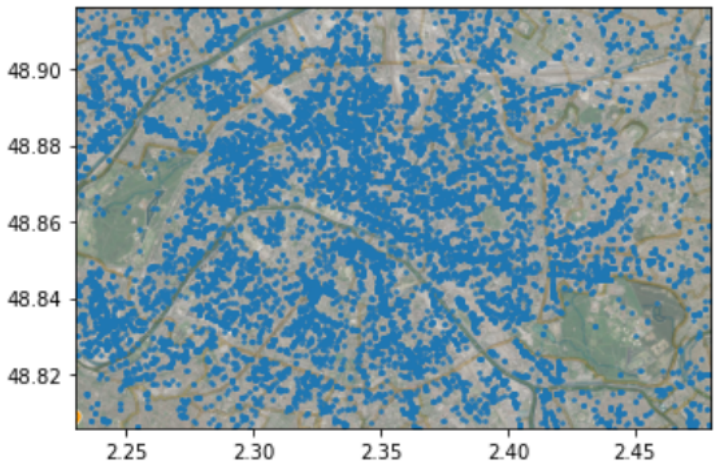
Avec l'aide de THAUVIN Dao & STERKERS Luc

1 Classe Density

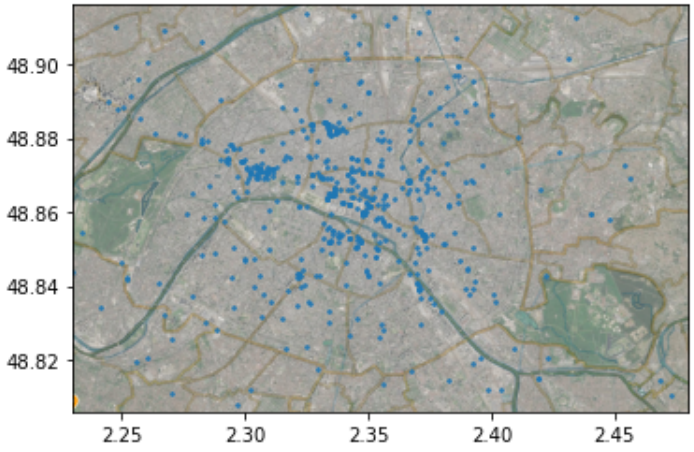
La méthode *score(data)* permet de renvoyer la log-vraisemblannce des données *data* de l'estimateur. Pour gérer les point de densité nulle, on ajoute une très petite valeurs de manière à ce que les valeurs données au log soient différentes de 0 car log(0) n'est pas défini.

2 Données : API Google places, Points d'intéret de Paris

Affichage des POIs de Paris :
Bars et restaurants *geo_mat*

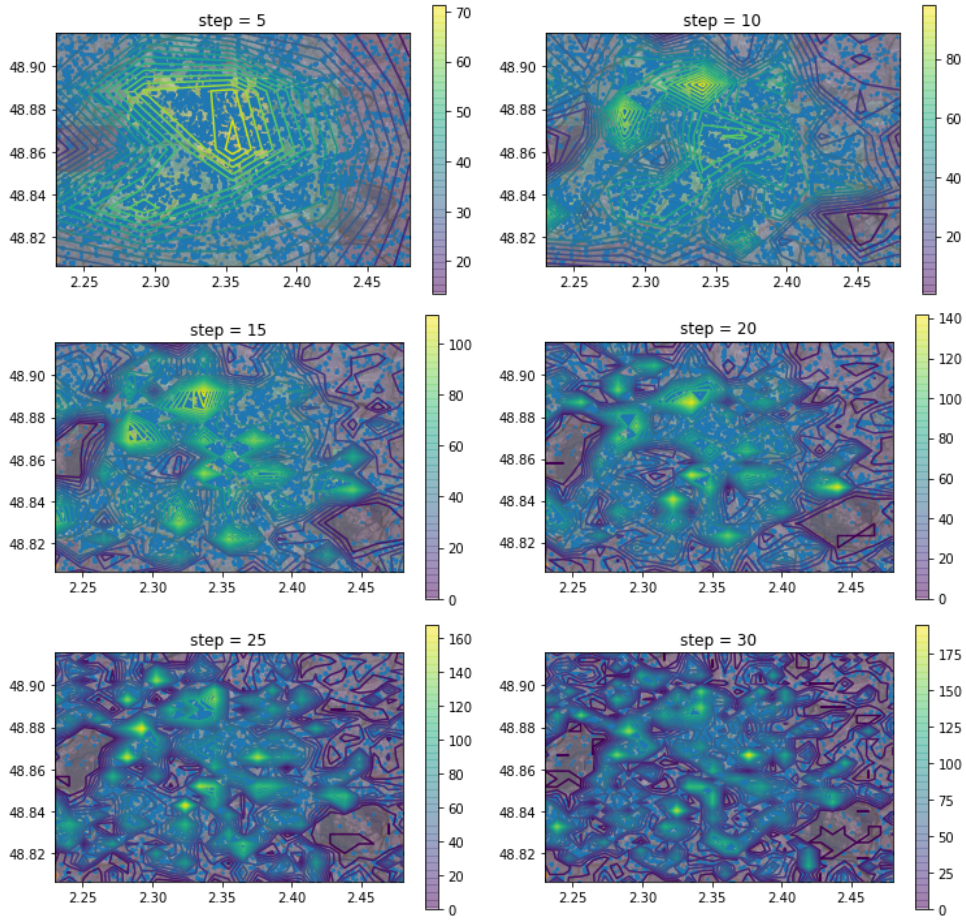


Boites de nuits *geo_night*



2.1 Méthode par histogramme

1. Résulats et différents pas de discrétisation



On voit clairement que l'on s'approche du sur-apprentissage quand le step est trop grand, visuellement on peux choisir un pas de 15/20 en dessous on risque d'être en sous apprentissage et au dessus en sur-apprentissage.

2. Vérification d'une loi de densité

Vérification que l'on obtient bien une loi de densité : Pour vérifier cela, on regarde que l'intégral de la densité sur notre espace soit égale à 1 :

$$\sum_i \sum_j V_{i,j} p(i,j) \simeq 1$$

Application :

$$\sum_i \sum_j ((xmax - xmin) * (ymax - ymin))/(20 * *2) * h.histogram[0][i,j] \simeq 1$$

Nous obtenons : *0.9999999999999999* nous avons donc bien une densité de probabilité.

3. Vérification de l'estimateur

Notre estimateur de densité est créer par la fonction *fit* qui permet de discretiser la grille et d'estimer la densité en chaque case. Si il n'y a pas d'erreur, notre estimateur sur la grille doit vérifier les résultats obtenus avec la fonction *get_ density2D* :

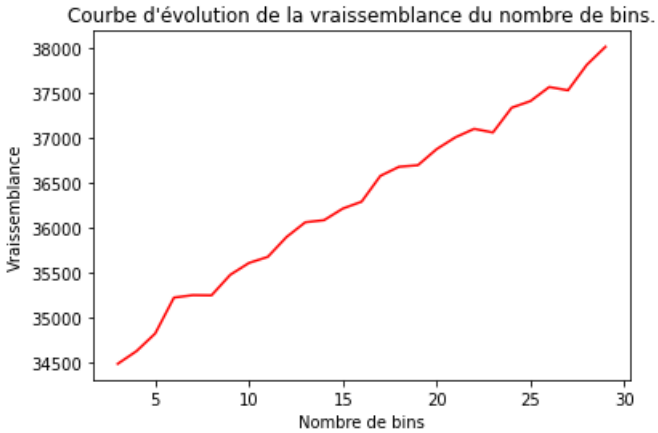
Nous obtenons les mêmes résultats pour l'estimation en chaque point de la grille et la discretisation de la grille sur le premier axe et le second.

1. Estimation en chaque point de la grille :
[[23.41667822 8.62719724 3.69737025 23.41667822 11.09211074
19.71930798 8.62719724 20.95176472 30.81141871 2.4649135
41.90352945 6.16228374 30.81141871 11.09211074 20.95176472
14.78948098 33.27633221 33.27633221 4.92982699 46.83335644]
...
[19.71930798 17.25439448 16.02193773 46.83335644 49.29826994
27.11404847 4.92982699 2.4649135 30.81141871 12.32456748
27.11404847 14.78948098 52.99564018 18.48685123 6.16228374
7.39474049 23.41667822 18.48685123 17.25439448 2.4649135]]

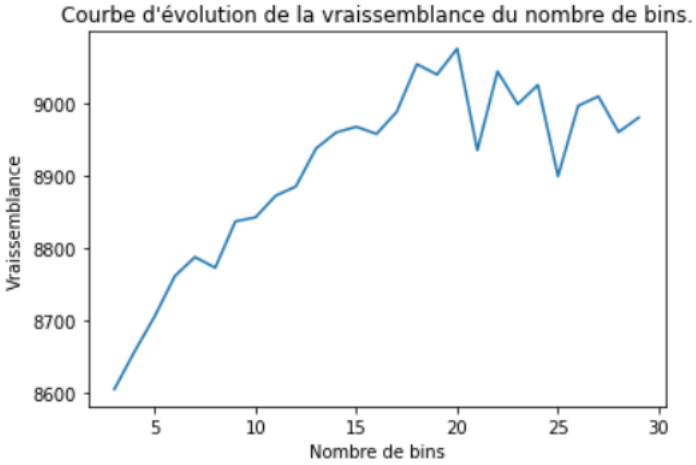
2. Discretisation sur le premier axe:
[2.2300034 2.24315825 2.25631309 2.26946794 2.28262279 2.29577764
2.30893248 2.32208733 2.33524218 2.34839703 2.36155187 2.37470672
2.38786157 2.40101642 2.41417126 2.42732611 2.44048096 2.45363581
2.46679065 2.4799455]

3. Discretisation sur le second axe:
[48.8060263 48.81181201 48.81759772 48.82338343 48.82916914 48.83495485
48.84074056 48.84652627 48.85231198 48.85809769 48.86388341 48.86966912
48.87545483 48.88124054 48.88702625 48.89281196 48.89859767 48.90438338
48.91016909 48.9159548]
Notre estimateur est alors correcte.

4. Split des données en ensembles d'apprentissage et de test
Les données sont séparées avec 20% en test et 80% en apprentissage. Dans un premier temps nous avons chercher quel pas de discrétisation semble le meilleur :
Ensemble d'apprentissage:



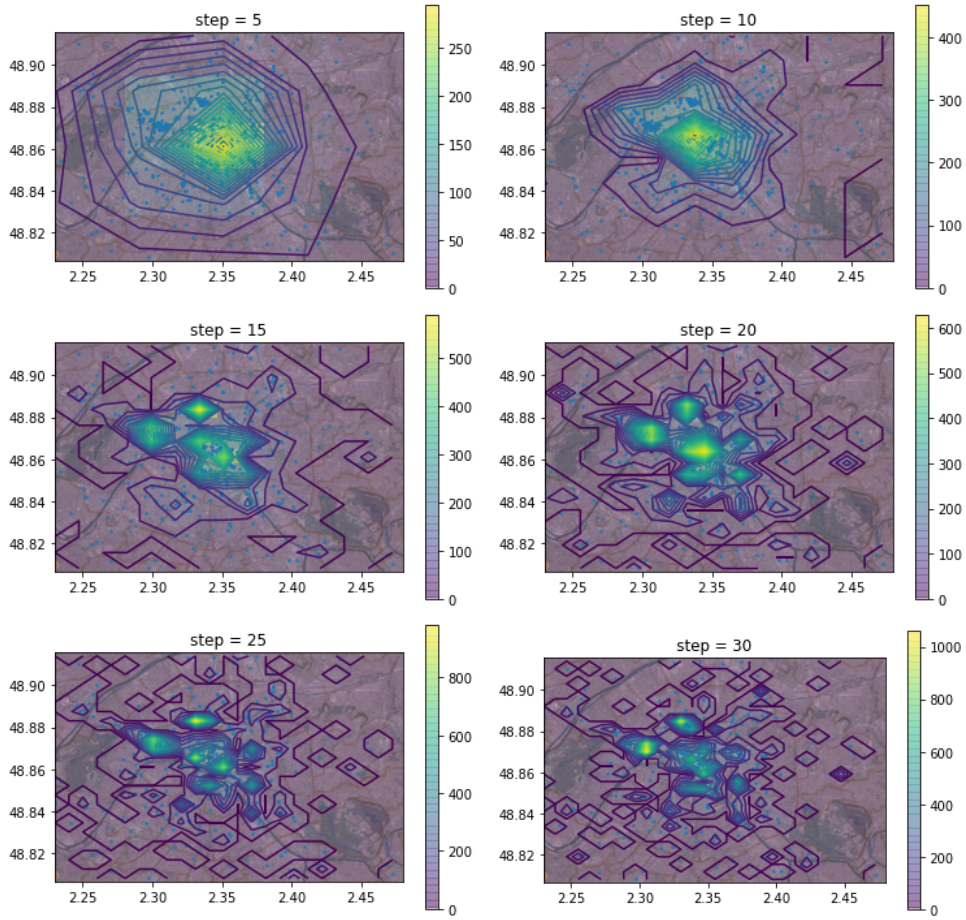
Ensemble de Test:



On peut remarque que sur les données d'apprentissage le score ne fait que augmenter, en effet le modèles apprend sur les données et évalue sur ces mêmes données, cependant sur l'ensemble de test on remarque qu'a partir d'un pas supérieur à environ 20, les performances diminue, en effet on est en sur-apprentissage, en dessous d'environ 20 on est en sous-apprentissage. Un pas de discrétisation au alentour de 20 semble être un bon compromis.

5. Expériences avec night_club

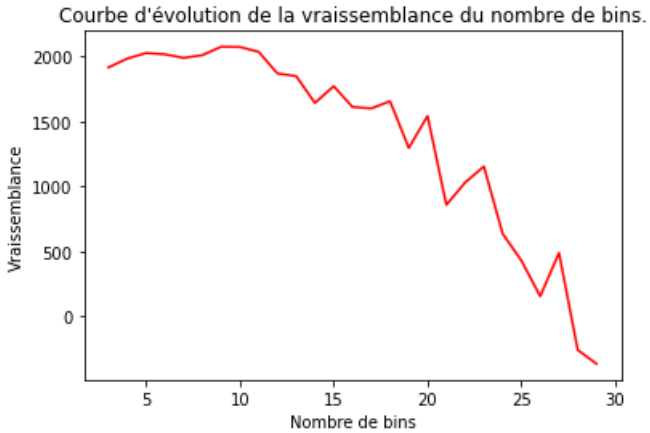
5.1. Résulats et différents pas de discrétisation



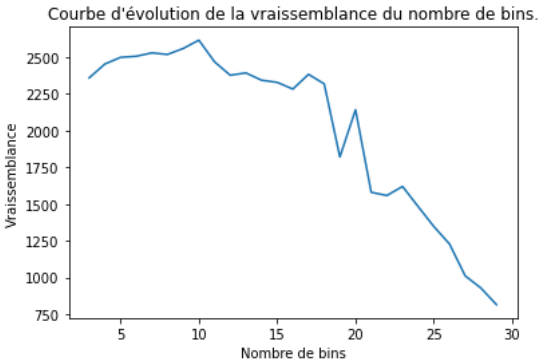
De même que précédemment il faut bien choisir le pas de discretisation pour éviter le sous et sur apprentissage. Ici visuellement on pourrait choisir un pas entre 15 et 20. De plus, comme il y a moins de donnée que sur le jeu de donnée *geo_mat* sur les bars et restaurants, le risque de sur-apprentissage est plus important, il faudra probablement choisir un pas plus faible.

5.2. Split des données en ensembles d'apprentissage et de test

Dans un premier temps nous avons chercher quel pas de discrétisation semble le meilleur :
Ensemble d'apprentissage:



Ensemble de Test:

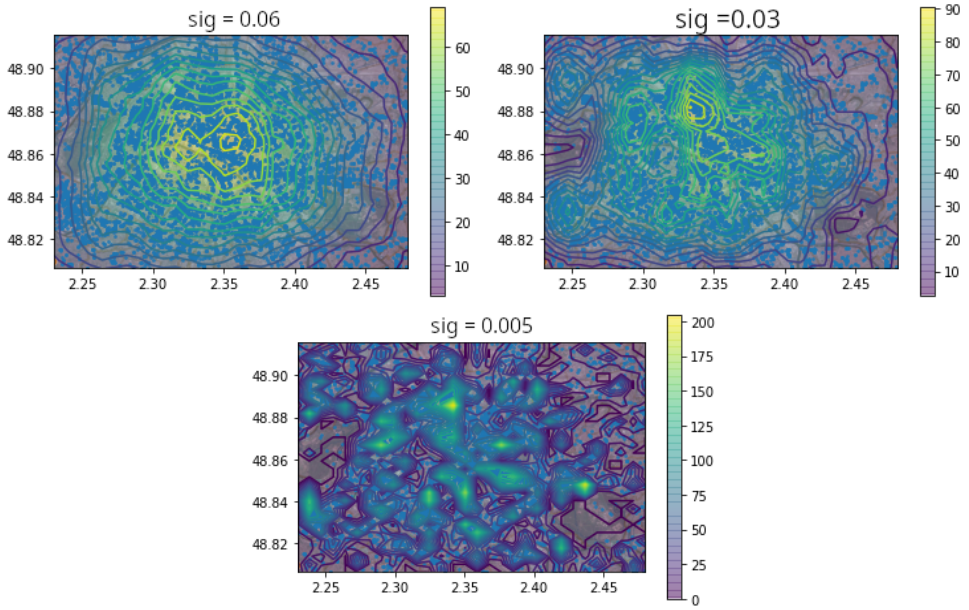


On peut remarquer que les scores des données de test comme d'apprentissage diminue très rapidement, à partir d'un pas de discrétisation de 3, ces valeurs sont suprenantent car le pas est très petit, cela peut être du au fait que les données de night club sont trop peu nombreuses et pas suffisemment représentativent pour être correctement exploitées. Dans cette situation, une validation croisée serait intéressante à mettre en place.

2.2 Méthode à noyaux

1. Noyaux Uniforme :

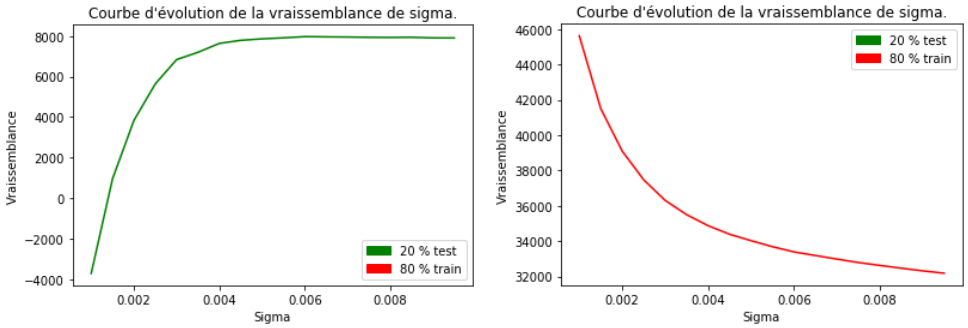
1.1 Résultats pour différentes valeurs de sigma:



On peut remarquer que plus les noyaux sont petits, plus on tends vers du sur-apprentissage, ilfaut tout de même faire attention et ne pas choisir un sigma trop grand pour ne pas tomber dans du sur-apprentissage.

1.2 Séparation Test/Train et Choix des hyperparamètres :
Dans un premier temps, on commence par chercher l'hyperparamètre sigma le plus performant, pour cela on compare les performances en test.

Courbe verte = test et Courbe rouge = train



On remarque que quand sigma augmente notre vraisemblence en test diminue, en effet plus sigma est grand plus la fenêtre est grande moins les résultats seront précis et donc plus la vraisemblence par raport aux résultats de train seront éloignés. On choisi alors un sigma proche de 0.006 car la vraisemblence est maximale en ce point.

1.3 Vérification d'une loi de densité

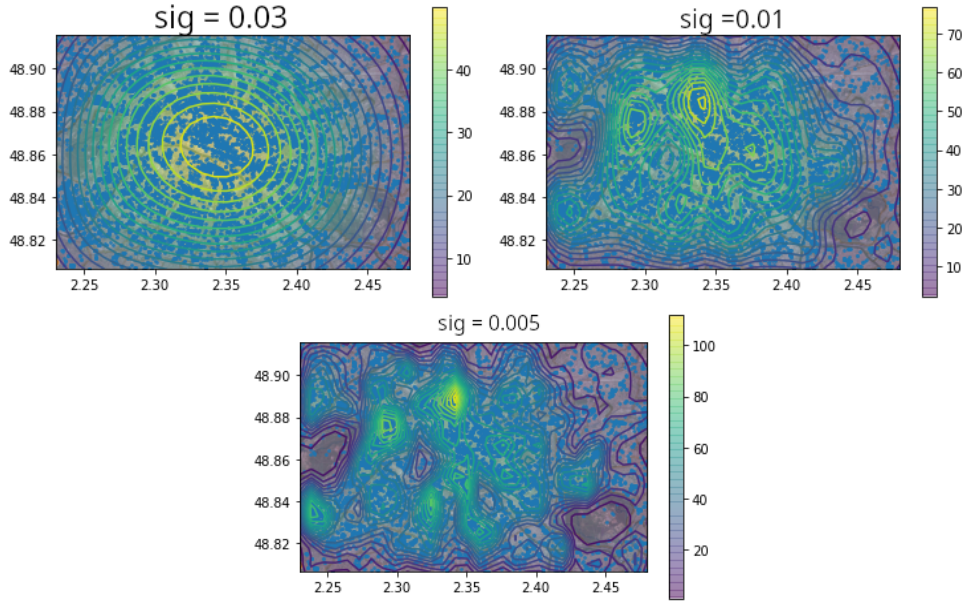
Pour vérifier cela, on utilise la fonction *get_ density2D* fournis :

$$\sum (res) * ((xmax - xmin) * (ymax - ymin))/(steps * *2) \simeq 0.94$$

Nous obtenons bien une densité car le résultat est proche de 1 (avec steps=30 et sigma=0.006). On utilise la fonction *get_ density2D* qui récupère la densité de rectangles en récupérant la densité d'un point. Pour l'histogramme, la densité d'un point correspond à la densité du rectangles ou il se situe, on avait donc un très bon résultat car *get_ density2D* fait le calcul de densité sur le même rectangle. Les noyaux ne sont pas forcement rectangle ce qui peut créer une marge d'erreur (notamment avec le noyau gaussien. Même avec le noyau uniforme, il est impossible de faire la même chose que pour la méthode par histogramme car nos noyaux sont carrés et non des rectangles.). Il faut aussi avoir une taille de noyau proche de la taille du rectangle de *get_ density2D* pour avoir une valeur proche de 1 (sinon des zones ne seront pas pris en compte ou pris en compte plusieurs fois).

2. Noyaux Gaussien :

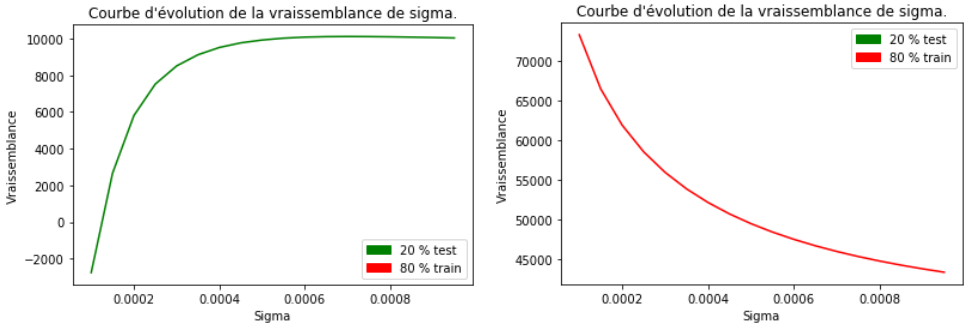
2.1 Résultats pour différentes valeurs de sigma:



On peut remarquer que plus les noyaux sont petits, plus on tends vers du sur-apprentissage, ilfaut tout de même faire attention et ne pas choisir un sigma trop grand pour ne pas tomber dans du sur-apprentissage.

2.2 Séparation Test/Train et Choix des hyperparamètres :
Dans un premier temps, on commence par chercher l'hyperparamètre sigma le plus performant, pour cela on compare les performances en test.

Courbe verte = test et Courbe rouge = train



On remarque que comme précédemment la vraissemblence en train diminue quand sigma augment pour les mêmes raison. On remarque cependant que la vraisemblance est maximale pour sigma = 0.0006 environ, la taille du kernel doit être plus petite avec un kernel Gaussien que Uniforme pour avoir des résultats comparable.

2.3 Vérification d'une loi de densité
Le calcul est le même que précédemment, nous obtenons 0.92

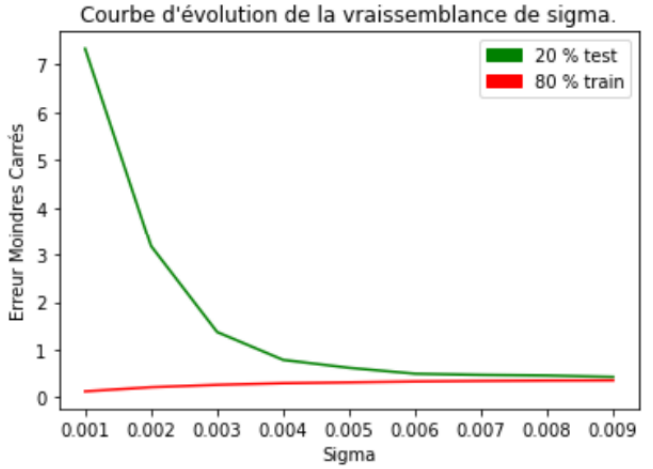
2.3 Régression par Nadaraya-Watson

La regression par Nadaraya-Watson permet de mettre en place une classification binaire, nous allons évaluer la qualité de la préciction à travers les predictions du modèle de nadaraya-Watson et les données de *train* déjà étiquettées par la méthode des moindres carrés.

L'estimateur est définie par : $f(x) = \sum_{i=1}^{n} \frac{y_i \phi\left(\frac{x - x_i}{\sigma}\right)}{\sum_{i=j}^{n} \phi\left(\frac{x - x_j}{\sigma}\right)}$

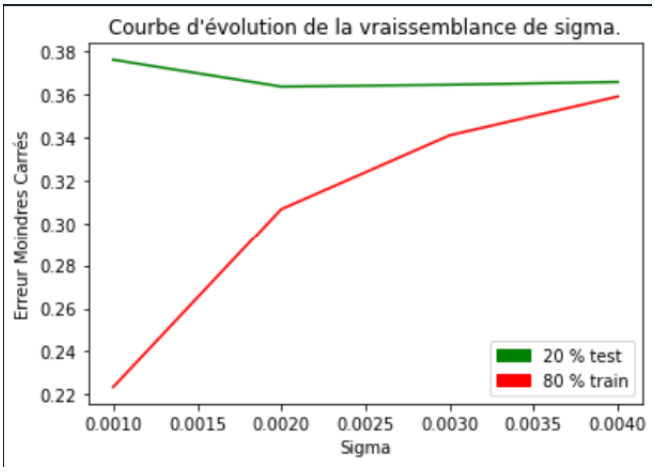
2.3.1 Performance et erreurs au sens des moindres carrés

Kernel Uniforme



On choisira donc un sigma envion égale à 0.006 pour le kernel uniforme.

Kernel Gaussien



On choisira donc un sigma envion égale à 0.002 pour le kernel gaussien.

2.3.2 Prediction de la note d'un POI en fonction de son emplacement

Kernel Uniforme avec $\sigma = 0.006$

- Les 10 premieres predictions :
  [4.04074074 3.95454545 3.86363636 3.71666667 3.82413793 3.8375 3.83809524 4.03 3.16153846 4.13333333]
- Les 10 réelles étiquettes (notes) :
  [4.1 4.7 3.5 3.5 2.4 4.3 4.3 3.4 3.7 3.8]
- Moyenne des 10 premières prédictions :
  [3.94642857]
- Moyenne des 10 premières notes réelles :
  [3.7699999999999996]

Kernel Gaussien avec $\sigma = 0.002$

- Les 10 premieres predictions :
  [4.06418109 3.96236548 3.95151621 3.63802376 3.87549742 3.94724 3.81388183 4.03524526 3.43808776 4.03212127]
- Les 10 réelles étiquettes (notes) :
  [4.1 4.7 3.5 3.5 2.4 4.3 4.3 3.4 3.7 3.8]
- Moyenne 10 premières des prédictions :
  [3.7699999999999996]
- Moyenne des 10 premières notes réelles :
  [3.7699999999999996]

On remarque que les prédictions semble plutôt bonnes dans les 2 cas, le kernel gaussien possède un temps de calcul un peu plus long que le Kernel uniforme malgré tout.