Intelligence Artificielle & Apprentissage

Compte-Rendu de TPs

5A IA2R SIR – Baptiste STOLL

05/12/2021

Sommaire

PARTIE A - TP N°1

Discrimination multi-classes : Reconnaissance de caractères manuscrits par l'algorithme des k plus proches voisins

I - Introduction

Dans ce TP (et les suivants) nous allons utiliser la librairie Python **sklearn**.

```
from sklearn import module
```

La procédure aura donc toujours la même structure globale :

1. Création du modèle :
   ```
   model = module.méthode(params…)
   ```
2. Apprentissage :
   ```
   model.fit(X, Y)
   ```
3. Prédiction :
   ```
   model.predict(Xtest)
   ```

II - Test de l'algorithme dans le plan

II.a. - Classification binaire linéairement séparable

2. En procédant ainsi, la séparation entre les deux zones s'apparente à une droite au centre des deux points centraux de chaque groupe de couleurs (médiatrice).

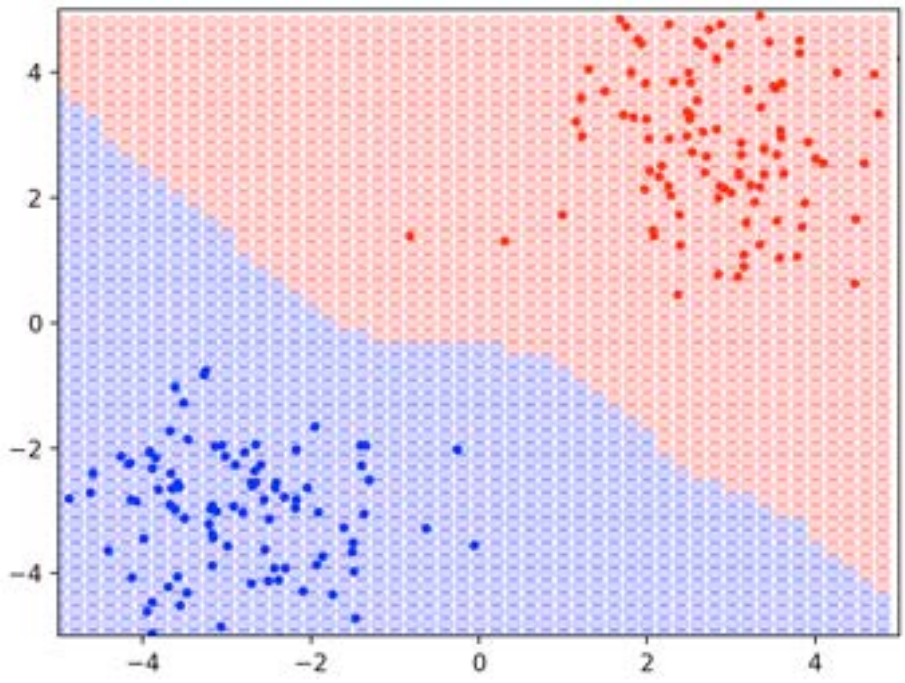

Figure 1 : Graphique pour bleus (-3,-3) avec variance = 1 et rouge (3,3) avec variance = 1

3. En plaçant cette fois-ci 200 point bleu avec un variance de 1 et 200 points rouges avec une variance de 4, la séparation entre les deux groupes se rapproche d'une parabole entourant le groupe de plus faible variance.

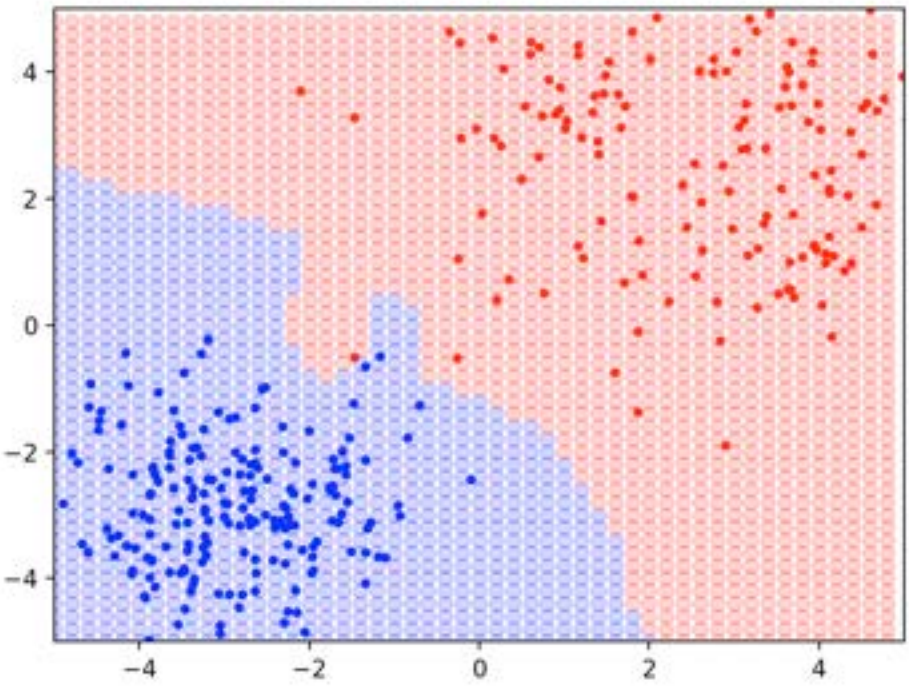

Figure 2 : Graphique pour bleus (-3,-3) avec variance = 1 et rouge (3,3) avec variance = 4

Notons ici qu'un point rouge s'est fortement éloigné de sa zone provoquant ainsi un creux marqué à l'intérieur de notre zone bleu. Ceci serait moins visible avec un plus grand nombre de points.

II.b. - Cas non séparables

Plus k est grand, plus les frontières entre les groupes sont « moyennées », un petit k va créer des frontière précises et très réparties. À l'inverse, un k très grand va regrouper au maximum les zones jusqu'à totalement en supprimer certaines pour un k trop grand.

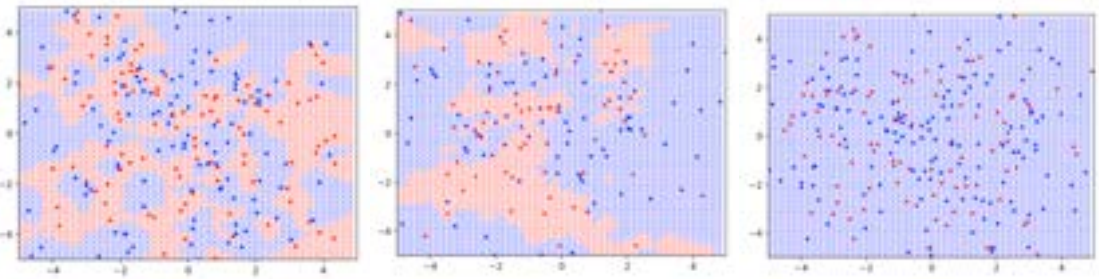

Figure 3 : Différents graphiques avec K=1, K=10 et K=100

II.c. - Problèmes multi-classes

Nous obtenons des résultats similaires mais cette fois-ci adaptés à plusieurs catégories. Par exemple ici avec plusieurs « droites » qui s'intersectent pour les frontières.

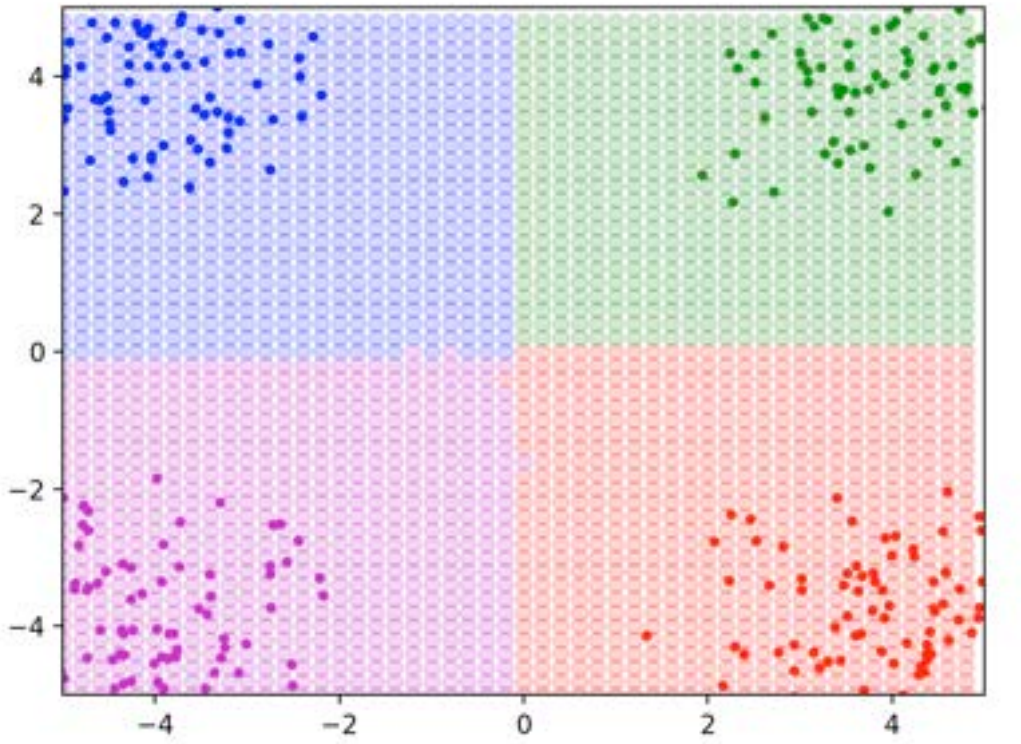

Figure 4 : Graphiques pour 4 catégories et 100 points par catégorie (variance de 1)

III - Application à la reconnaissance de caractères

III.a. - Base de données MNIST

1. $nb_{images} \times nb_{pixel} \times 8 + nb_{images} = nb_{images} \times 8(nb_{pixel} + 1)$

$= 10000 \times 8(784 + 1) \cong 62\ Mo$

2-3. Pour K=1, on obtient une erreur de généralisation d'environ 5%.

4. Avec notre jeu de données, nous trouvons que K=3 est la valeur produisant le moins d'erreur (à peu de choses près). Notons que lorsque l'on relance le programme, les données choisies changent et donc la meilleure valeur de K peut ne pas être la même du fait de la faible différence entre chaque erreur pour un $K \in [1, 10]$ différent.

```
Taille de la base d'apprentissage :  (10000, 784)
K=1, Erreur=0.051
K=2, Erreur=0.059
K=3, Erreur=0.047
K=4, Erreur=0.049
K=5, Erreur=0.05
K=6, Erreur=0.054
K=7, Erreur=0.055
K=8, Erreur=0.058
K=9, Erreur=0.058
K=10, Erreur=0.057
```

Figure 5 : Résultats des erreurs pour différents valeurs de K

5. En comptant le nombre d'erreur par couple de chiffres, on trouve qu'un certain nombre de couples sont confondus plus souvent que d'autres, par exemple sont souvent confondus :

- 3 et 5
- 3 et 8
- 5 et 8
- 2 et 7
- 7 et 9
- 8 et 9
- 1 et 2
- etc…

On remarque que les couples souvent confondus ont généralement des similitudes et peuvent fortement se ressembler. En effet, certains chiffres peuvent laisser un doute même pour un humain, il est donc tout à fait compréhensible qu'un algorithme ne puisse être parfait sur ce type de reconnaissance.

6. Pour correctement évaluer l'erreur pour notre K choisi on sépare nos données en 3, une base d'apprentissage, une base de validation et une base de test. On choisit le meilleur K sur la base de validation puis on évalue sa réelle erreur sur notre base de test.

```
Taille de la base d'apprentissage :  (11000, 784)
K=1, err=0.061
K=2, err=0.071
K=3, err=0.061
K=4, err=0.06
K=5, err=0.059
K=6, err=0.06
K=7, err=0.067
K=8, err=0.069
K=9, err=0.071
K=10, err=0.071
Le K choisi est K=5 pour une erreur SUR LA BASE DE VALIDATION err=0.059.
Le meilleur K est K=5 pour une erreur RÉELLE de err=0.047.
```

Figure 6 : Valeur de K + son erreur réelle

7-8. Pour K=5 avec un apprentissage sur 60 000 cas et un test sur 10 000 cas, le temps de calcul est légèrement plus long (mais reste rapide, de l'ordre de la dizaine de secondes) et l'erreur est bien plus faible (de l'ordre de 2,7% d'erreur environ) ce qui est normal étant donné la base d'apprentissage bien plus conséquente. On remarque d'ailleurs que les chiffres fréquemment confondus restent sensiblement les mêmes que cités précédemment.

```
Pour K=5 sur un apprentisage sur 60 000 cas et un test sur 10 000 cas, on trouve une erreur de err=0.0271.
err : quantité
1-9 : 2
5-8 : 18
2-8 : 12
7-9 : 14
3-5 : 20
3-9 : 14
4-6 : 2
4-9 : 26
4-7 : 2
3-7 : 7
2-3 : 8
3-8 : 11
0-5 : 5
1-7 : 8
4-8 : 9
2-4 : 4
1-5 : 3
1-4 : 9
0-6 : 8
0-9 : 3
6-8 : 5
2-6 : 4
0-1 : 1
1-2 : 5
1-8 : 10
0-8 : 6
0-2 : 3
2-7 : 19
```

Figure 7 : Apprentissage, test sur la base complète (60 000 et 10 000) et liste du nombre d'erreurs par couples de chiffres.

Pour calculer la mémoire qu'occupe l'apprentissage sur la base complète, nous pouvons reprendre le calcul de la question 1 :

$$nb_{images} \times nb_{pixel} \times 8 + nb_{images} = nb_{images} \times 8\left(nb_{pixel} + 1\right) = 60000 \times 8(784 + 1) \cong 376{,}8\ Mo$$

PARTIE B - TP N°2

Classification SVM, méthodes de décomposition et classification de défauts de rails

I - Classification SVM

I.a. - Cas séparables

2. Voici la séparation des points par tracé de points colorés (similaire au TP n°1) :

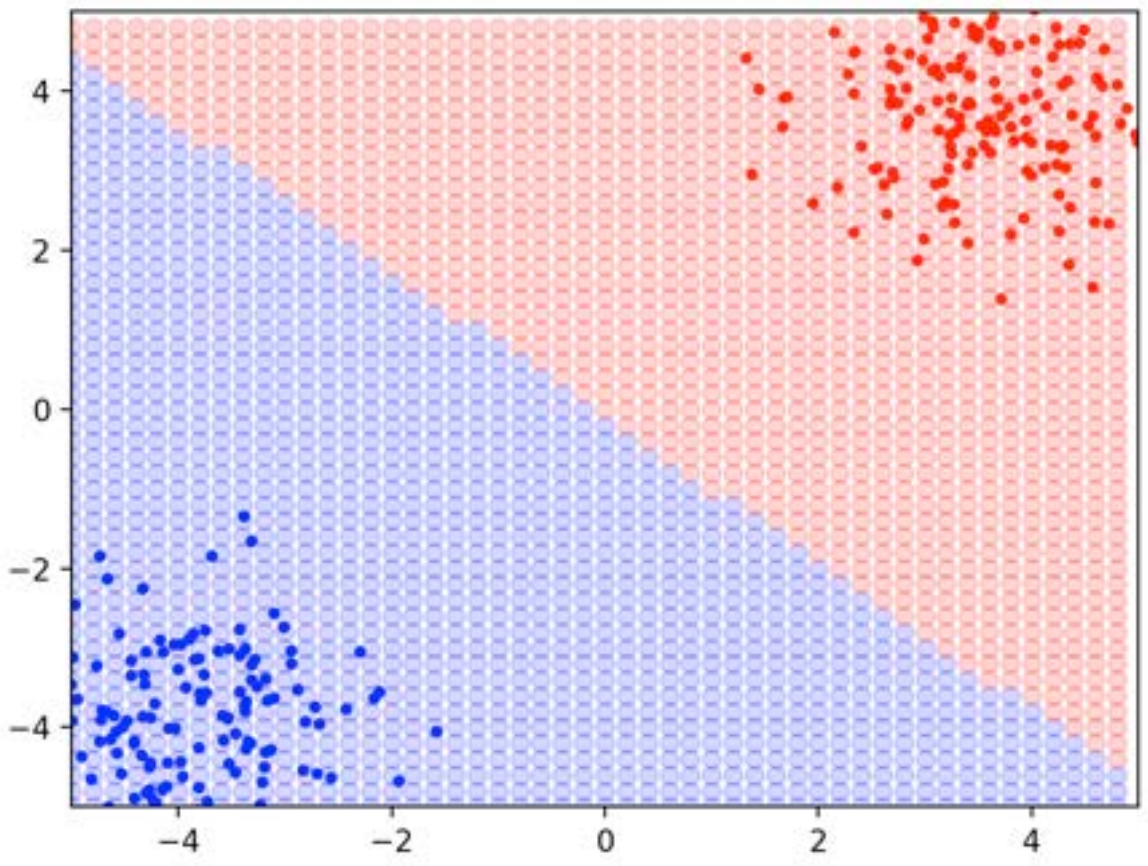

Figure 8 : Graphique par tracé d'une grille de points de test.

3. Avec $w = (w_1, w_2)$ et b, pour un x_1 donné, on peut retrouver x_2 avec :

$$w_1 \times x_1 + w_2 \times x_2 + b = 0$$

$$x_2 = \frac{-b - w_1 x_1}{w_2}$$

Pour pouvoir obtenir 2 points de la droite de séparation que l'on peut ensuite tracer.

On peut de même calculer deux points pour chaque frontière de Δ avec :

$$x_{2_{MargeSup}} = \frac{-b - 1 - w_1 x_1}{w_2}$$

$$x_{2_{MargeInf}} = \frac{-b - 1 - w_1 x_1}{w_2}$$

On peut ainsi tracer sur le graphe :

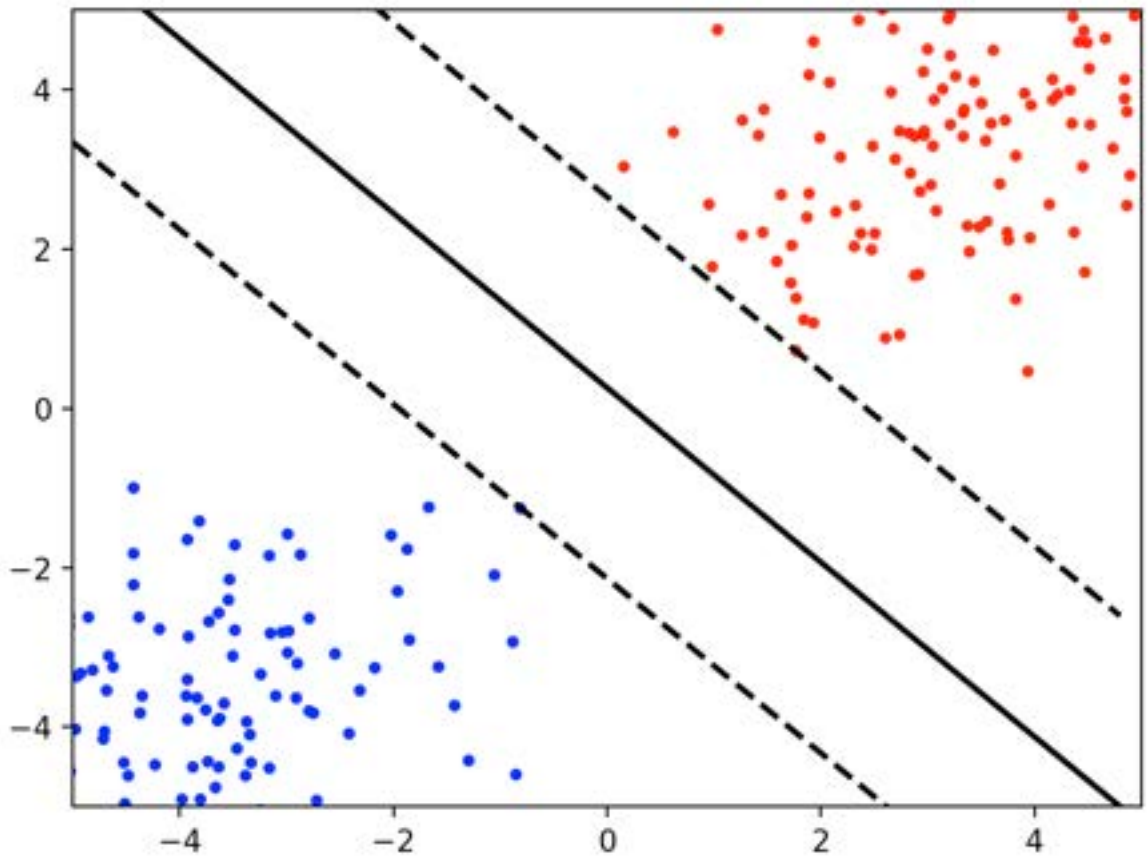

Figure 9 : Tracé de la séparation de de la marge Delta sur le graphe.

I.b. - Cas non séparables

On crée un jeu de données non linéairement séparable puis l'on affiche la séparation pour deux C différents (un C faible et un C très grand) :

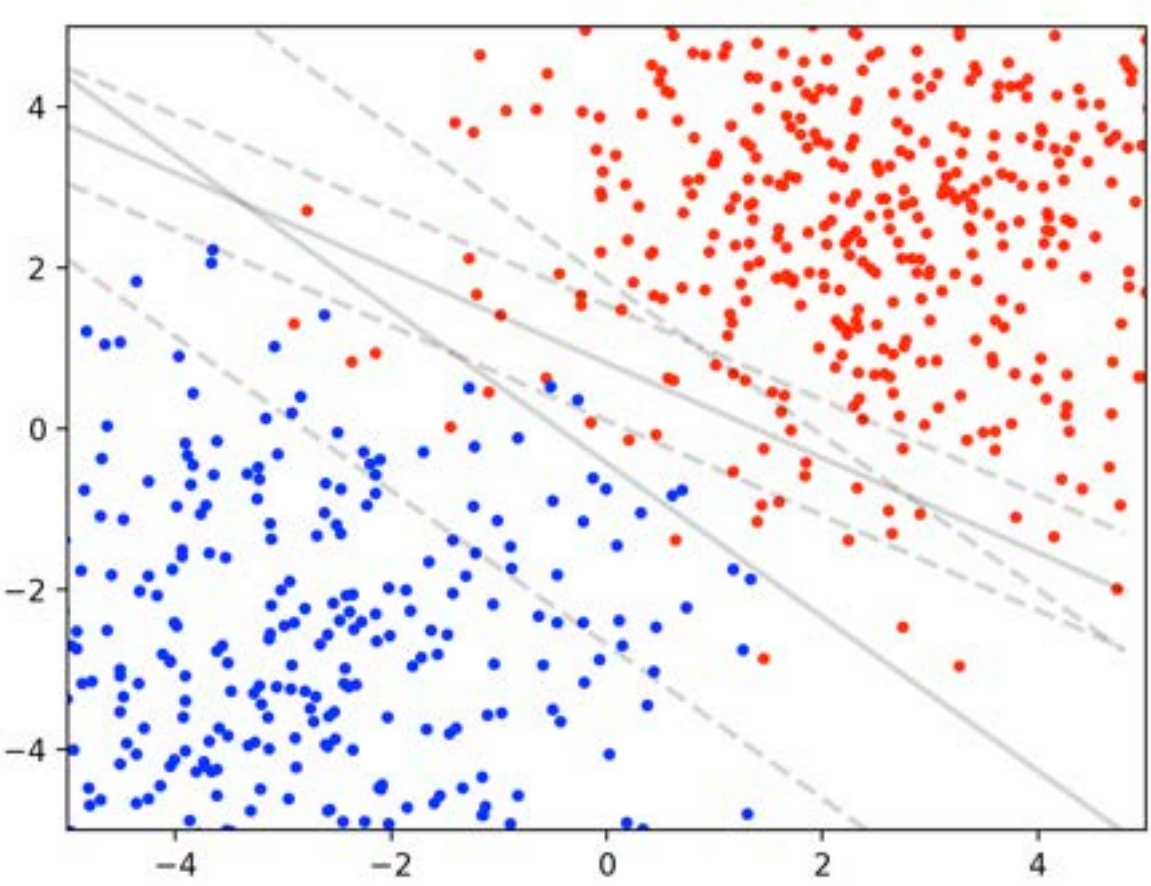

Figure 10 : Graphique de points non linéairement séparables pour deux valeurs de C.

On constate non seulement que les deux droites de séparation ne sont pas les mêmes pour une valeur de C différente mais aussi que pour un C très grand, la marge Δ est plus faible (car on accorde plus d'importance aux erreurs).

I.c. - Classification non linéaire

Voici le support model pour différentes valeurs de C et de sigma :

- 160 points, C = 0.25 et sigma = 0.5
  - model support = [72 68]
- 160 points, C = 0.25 et sigma = 1
  - model support = [43 40]
- 160 points, C = 0.25 et sigma = 2
  - model support = [19 18]
- 160 points, C = 1 et sigma = 0.5
  - model support = [56 53]
- 160 points, C = 1 et sigma = 1
  - model support = [29 33]
- 160 points, C = 1 et sigma = 2
  - model support = [10 11]
- 160 points, C = 32 et sigma = 0.5
  - model support = [56 53]
- 160 points, C = 32 et sigma = 1
  - model support = [29 30]
- 160 points, C = 32 et sigma = 2
  - model support = [8 8]

Pour des groupes de points bien séparés, voici l'impact de variation de C et de sigma :

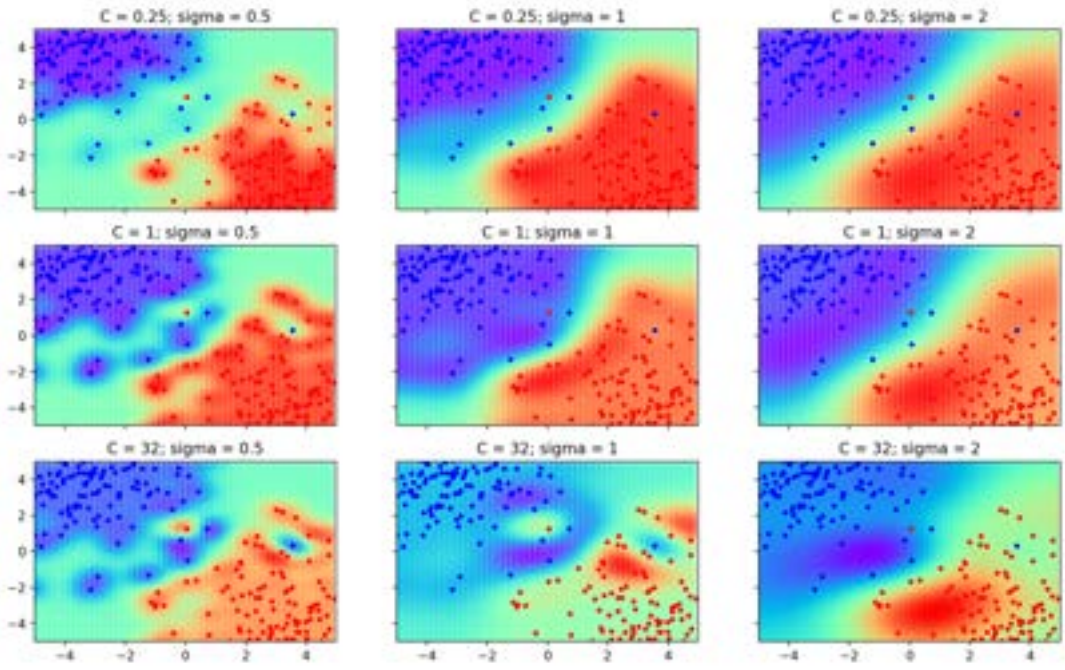

Figure 11 : Affichage de la fonction de séparation pour différentes valeurs de C et de sigma.

On peut faire de même pour des points « entourés » par un autre groupe de points :

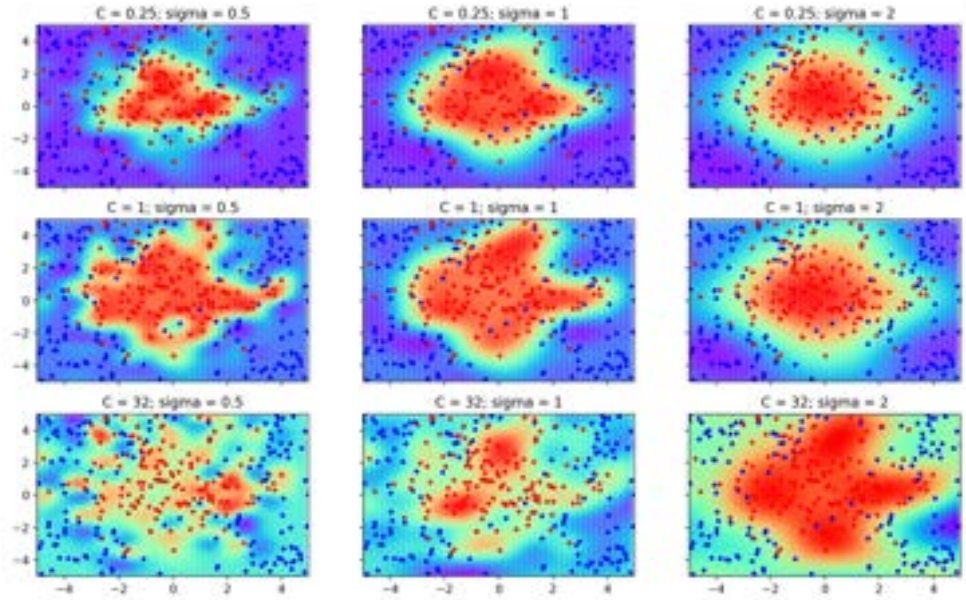

Figure 12 : Affichage de la fonction de séparation pour différentes valeurs de C et de sigma.

Idem pour des points totalement mélangés :

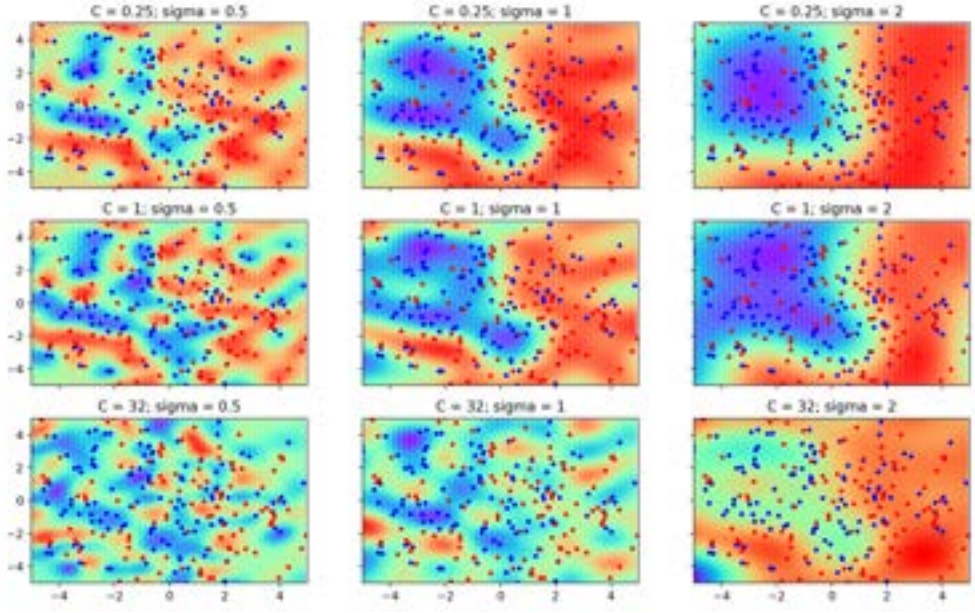

Figure 13 : Affichage de la fonction de séparation pour différentes valeurs de C et de sigma.

De façon générale, on constate que plus C est grand, moins la frontière entre les zones et nette (beaucoup de vert entre le bleu et le rouge) et plus la répartition semble « diffuse ». Un C faible produit une fonction plus « sûre » d'elle est donc réalisant potentiellement plus d'erreur. De même plus sigma et grand, plus la forme de séparation est « lissé » et colle moins précisément aux données d'apprentissage. À l'inverse un sigma faible produit un résultat très proche des points d'apprentissage.

II - Problème multi-classe et techniques de validation croisée : application à la classification de défauts de rails

II.a. - Classifieurs binaires

```
# Création des classifieurs binaires et évalutation de l'erreur sur la base d'apprentissage.

Cas 1 contre tous : erreur d'apprentissage = 0.014285714285714285
Cas 2 contre tous : erreur d'apprentissage = 0.007142857142857143
Cas 3 contre tous : erreur d'apprentissage = 0.0
Cas 4 contre tous : erreur d'apprentissage = 0.0
```

Figure 14 : Screen du résultat de notre programme pour l'évaluation des erreurs d'apprentissage pour chaque cas "i contre tous".

Pour chaque classifieurs binaires, on obtient une erreur très faible voir même totalement nulle.

II.b. - Combinaison des classifieurs binaires

Nous créons ensuite un classifieur multi-classes par combinaison des classifieurs binaires.

```
# Création du classifieurs multi-class général par combinaison des classifieurs binaires.

Erreur d'apprentissage sur le multi-class global = 0.0
```

Figure 15 : Screen de notre programme pour l'évaluation de l'erreur d'apprentissage sur notre modèle multi-class.

Pour ce classifieur, nous obtenons une erreur d'apprentissage nulle. Ce qui n'est pas très surprenant d'autant plus que le nombre d'éléments d'apprentissage est très faible.

II.c. - Estimation de l'erreur de généralisation par validation croisée

Une façon plus « fiable » d'évaluer la véritable erreur de notre modèle est la méthode par validation croisée. Nous allons donc successivement isoler chaque élément de notre base d'apprentissage, apprendre sur la base privée de cet élément et tester la prédiction de notre modèle sur cet élément isolé.

3.

```
L'erreur de généralisation par méthide LOO est éstimée à err = 0.07857142857142857
```

Figure 16 : Screen de notre programme pour l'évaluation de l'erreur par validation croisée sur notre modèle multi-class.

4-5. On peut pour chaque classifieurs binaire et pour chaque index allant de 0 à 139, déterminer si oui ou non il a correctement prédit notre valeur d'index i isolée. Nous représentons un « **O** » si la prédiction est bonne et une « **X** » sinon, on fait de même pour la prédiction finale (multi-class).

Pour chaque i (allant de 0 à 140), on affiche :

`[ index ] bin_1 bin_2 bin_3 bin_4 -> multi-class`

Figure 17 : Screen de notre programme pour le test sur la valeur d'index i pour i allant de 0 à 139.

On remarque que lorsqu'aucune erreur n'est faite lors des tests binaires, aucune erreur ne sera faite dans notre modèle multi-class. Une seule erreur pour nos modèle binaire peut causer une erreur sur le modèle multi-class mais ce n'est pas systématique. À partir de 2 erreurs, il y a de forte chance que le modèle multi-class se trompe (sauf cas du index = 115). Il est évident que la précision des modèles binaires a un impact non négligeable sur la précision finale du modèle multi-class.

On peut également compter le nombre d'erreurs réalisées par chaque classifieur binaire sur la prédiction de l'élément d'index i (isolé de l'apprentissage). On obtient :

```
Le classifieur binaire 1 a fait 15 erreur(s) sur 140 soit une erreur err = 0.10714285714285714
Le classifieur binaire 2 a fait 9 erreur(s) sur 140 soit une erreur err = 0.06428571428571428
Le classifieur binaire 3 a fait 5 erreur(s) sur 140 soit une erreur err = 0.03571428571428571
Le classifieur binaire 4 a fait 3 erreur(s) sur 140 soit une erreur err = 0.02142857142857143
```

Figure 18 : Screen de notre programme pour le nombre d'erreur sur la prédiction de i (de 0 à 140) pour chaque classifieur binaire.

On voit que les classifieurs **1** et **2** font le plus d'erreurs. Améliorer ces deux classifieurs binaires pourrait également améliorer les performances de notre modèle multi-class.

PARTIE C - TP N°3

Régression non linéaire et non paramétrique

I - Régression non linéaire

1-2. En affichant notre fonction de régression, nos 1000 points générés et notre base d'apprentissage (30 points), nous obtenons la figure suivante :

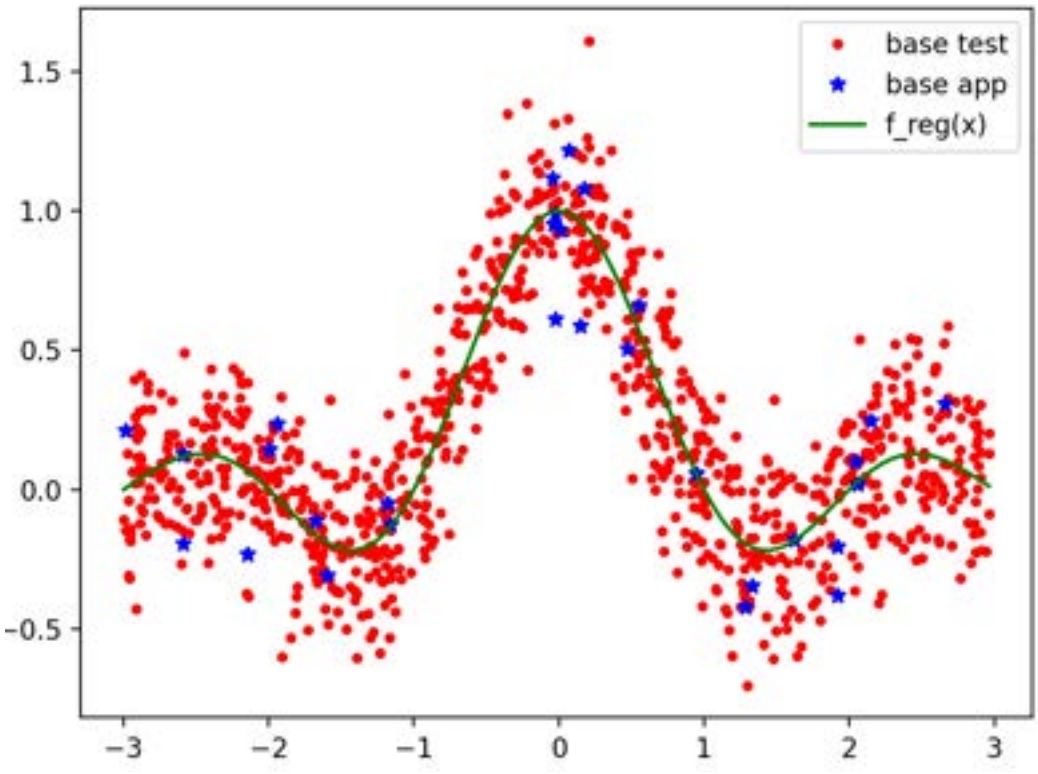

Figure 19 : Graphique de la fonction de régression, des points de test (rouge) et des points d'apprentissage (bleu).

II - Méthodes non paramétriques pour la régression non linéaire

II.a. - Kernel ridge regression

1-2. Nous testons notre méthode KRR sur nos données avec différentes valeurs pour sigma et lambda puis nous affichons l'ensemble des graphiques obtenus pour les comparer :

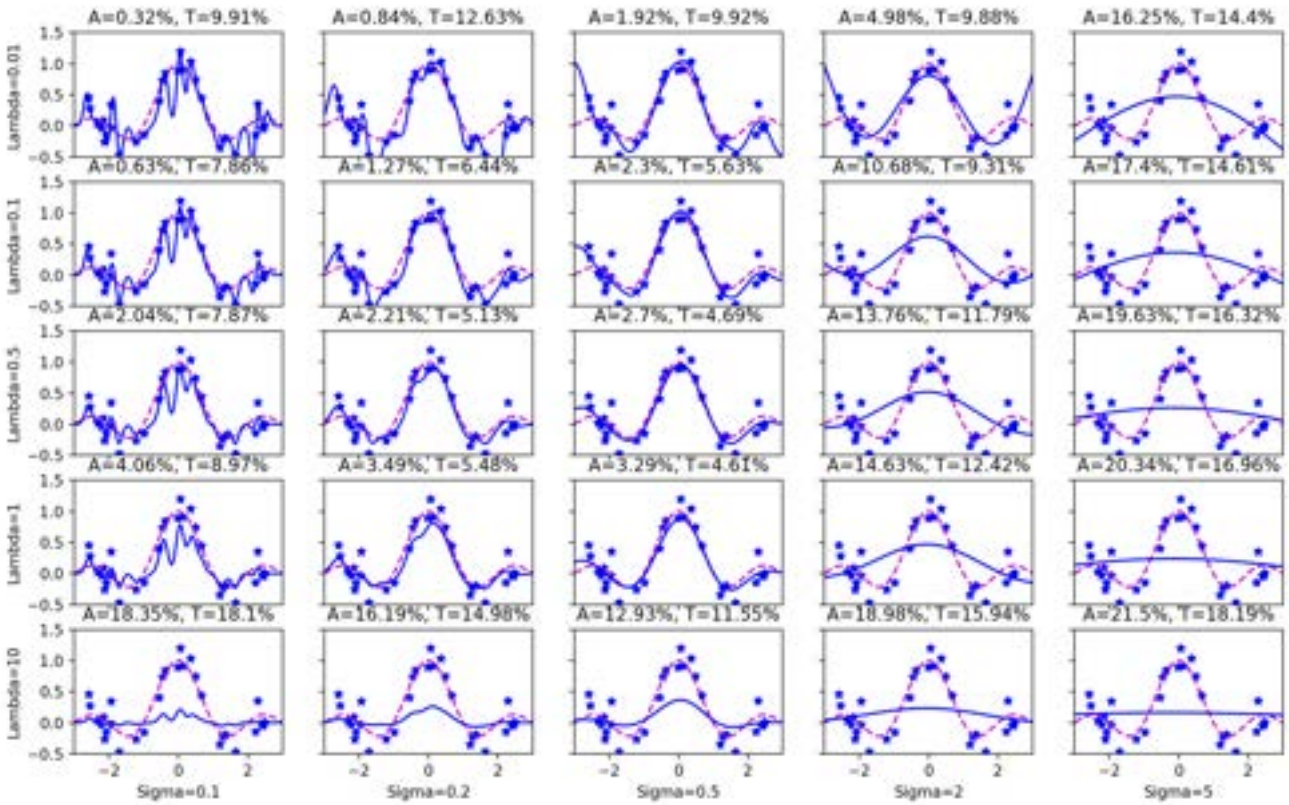

Figure 20 : Graphique obtenus pour différentes valeurs de Lambda et de Sigma, A et T sont respectivement les erreurs d'apprentissage et de test.

Nous pouvons donc identifier l'influence des paramètres Lambda et Sigma sur la qualité de la prédiction de notre fonction :

- Un sigma faible engendre une amplitude de sinusoïde plus faible, ce qui a tendance à faire beaucoup osciller notre fonction de prédiction. À l'inverse un sigma plus élevé va fortement augmenter l'amplitude de no sinusoïdes et produire un résultat beaucoup plus ample et lisse.
- Plus lambda est élevé, plus notre courbe est lissée, un grand lambda va effacer les irrégularités jusqu'à même totalement aplanir notre résultat s'il est trop grand. À l'inverse un lambda très faible laissera apparaitre de plus grandes irrégularités et une courbe moins lisse.

Par combinaison de ces deux hyper-paramètres, il semblerait que dans notre cas un lambda d'environ 1 et un sigma d'environ 0.5 fournissent des résultats plutôt satisfaisants, avec une erreur d'apprentissage d'environ 3.29% et une erreur sur la base de test d'environ 4.61%.

Nous remarquons au passage qu'une très faible erreur d'apprentissage n'est pas du tout signe de bon modèle avec par exemple une erreur d'apprentissage de 0.32% mais une erreur de test de presque 10% pour lambda = 0.01 et sigma = 0.1 (cas de sur-apprentissage).

3. En utilisant cette fois-ci le modèle KRR de la bibliothèque scikit-learn et en traçant la courbe obtenue sur notre graphique (cette fois-ci en rouge), on constate que les deux courbes se superposent parfaitement. Nous obtenons donc exactement le même résultat.

II.b. - K-plus proches voisins pour la régression

1-2. Nous complétons la fonction kppvreg puis nous traçons la prédiction obtenue pour différentes valeurs de K :

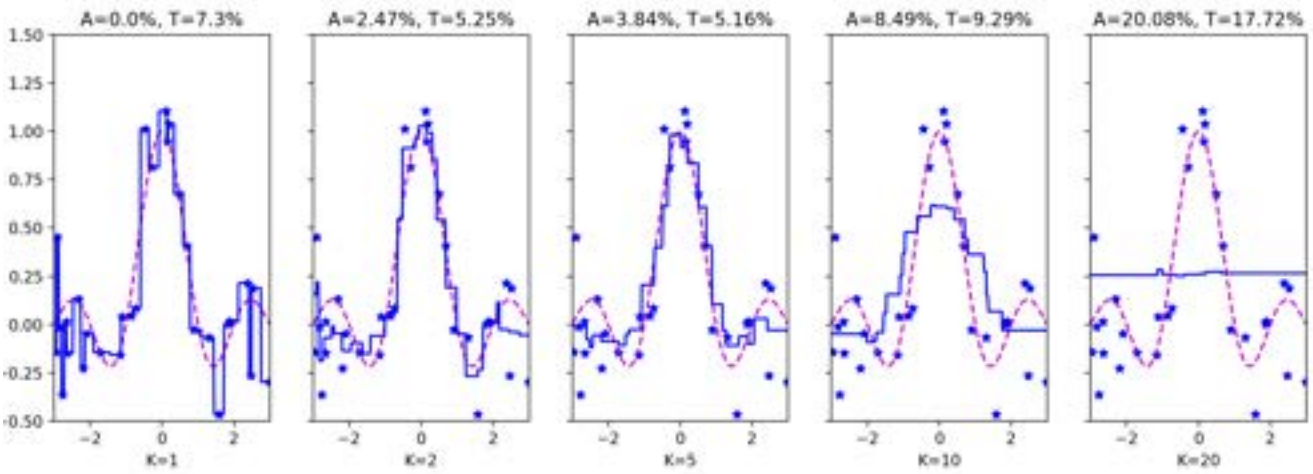

Figure 21 : Graphique des différentes prédictions par méthode des Kppv pour différents valeurs de K.

Par méthode des Kppv, nous obtenons des résultats tout de même satisfaisants pour une valeur de K = 5 (5.16% est une erreur plutôt honorable pour la simplicité de la méthode). Une valeur trop faible aura tendance à favoriser un sur-apprentissage (erreur d'apprentissage nulle mais erreur de test conséquente) et une valeur trop élevée aura tendance à trop moyenner le résultat (plus K se rapproche du nombre de point d'apprentissage plus la prédiction tend vers une simple constante ayant pour valeur la moyenne des points).

3. De même que dans la partie précédente, nous traçons en rouge la prédiction faite par scikit-learn et une fois de plus nous obtenons exactement les mêmes résultats.

PARTIE D - TP N°4

K-means et spectral clustering

I - Test de l'algorithme dans le plan

I.a. - K-means

1-3. Nous complétons la fonction puis testons sur deux groupes de point bien séparés pour différentes valeurs de K :

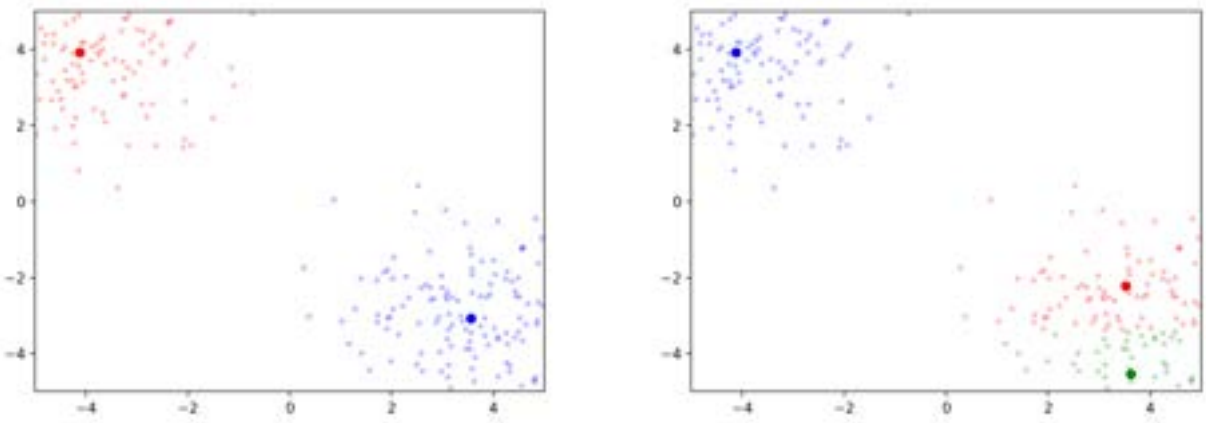

Figure 22 : Représentation graphique des résultats pour K = 2 et K = 3.

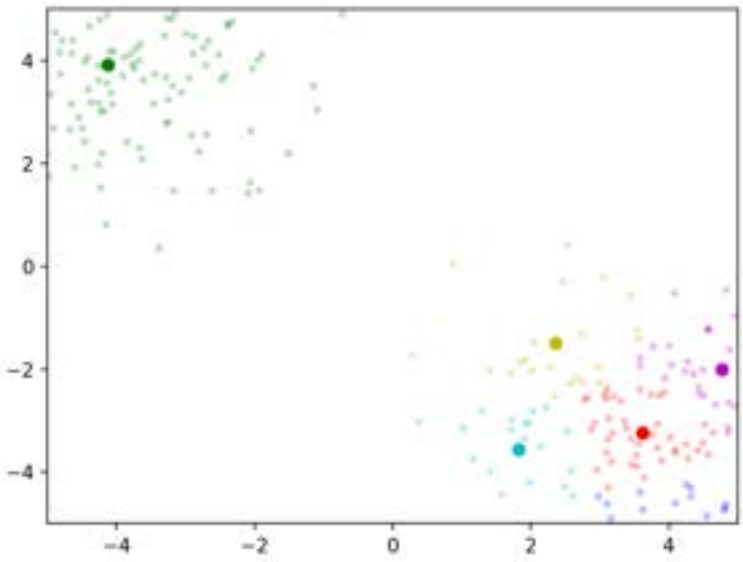

Figure 23 : Représentation graphique des résultats pour K = 6.

On remarque que lorsque K est égale au nombre de groupes distincts créés, l'algorithme se débrouille plutôt bien pour les identifier. Cependant dès lors que K est plus grand alors les groupes se retrouvent arbitrairement scindés en plusieurs sous-groupes de façon quasi aléatoire.

4. Lorsque les points sont moins bien séparés, l'algorithme semble couper l'ensemble de point aléatoirement en K parties plus ou moins égales. On remarque de plus qu'à chaque lancement de l'algorithme, le résultat est très différent du précédent. Enfin, une limite évidente de cette algorithme est l'incapacité à différencier des groupes non convexes, même lorsque ceux-ci sont bien séparés.

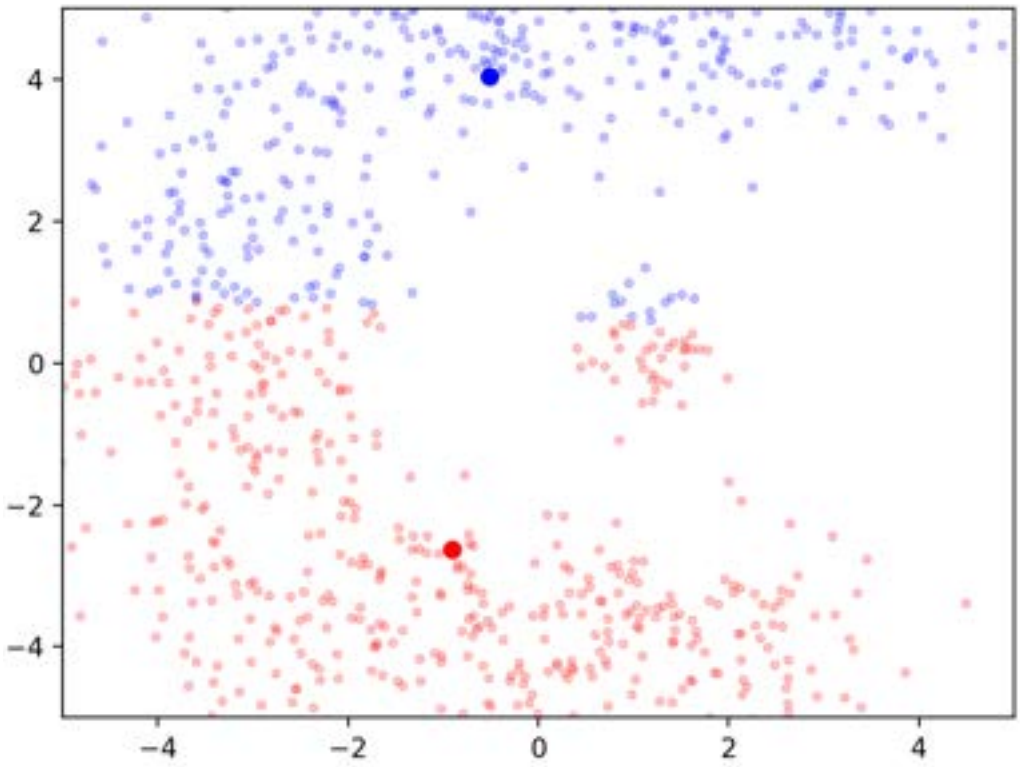

Figure 24 : Exemple de groupes convexes mal différenciés par l'algorithme K-means.

5. En procédant par centaine d'initialisations, l'aspect aléatoire est fortement réduit, mais le problème cité précédemment persiste. L'exécution avec la distance la plus faible entre chaque centre de groupe et ses points sera choisi comme la meilleurs solution, en effet cet algorithme fournit un optimum local et ne garantit pas avec une unique exécution qu'il s'agisse bien de la solution optimale globalement.

I.b. - Spectral clustering

1-2. Ce second algorithme semble fournir de bien meilleurs résultats que le précédents dans les cas complexes tels que des groupes de forme convexe.

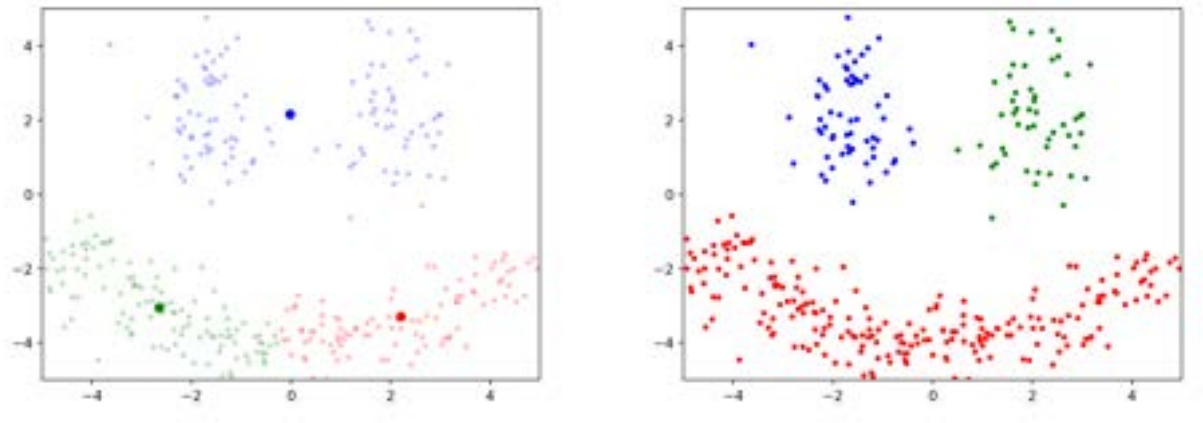

Figure 25 : À gauche, le résultat fournit par K-means, à droite celui fournit par le spectral-clustering (K=3, sigma=1).

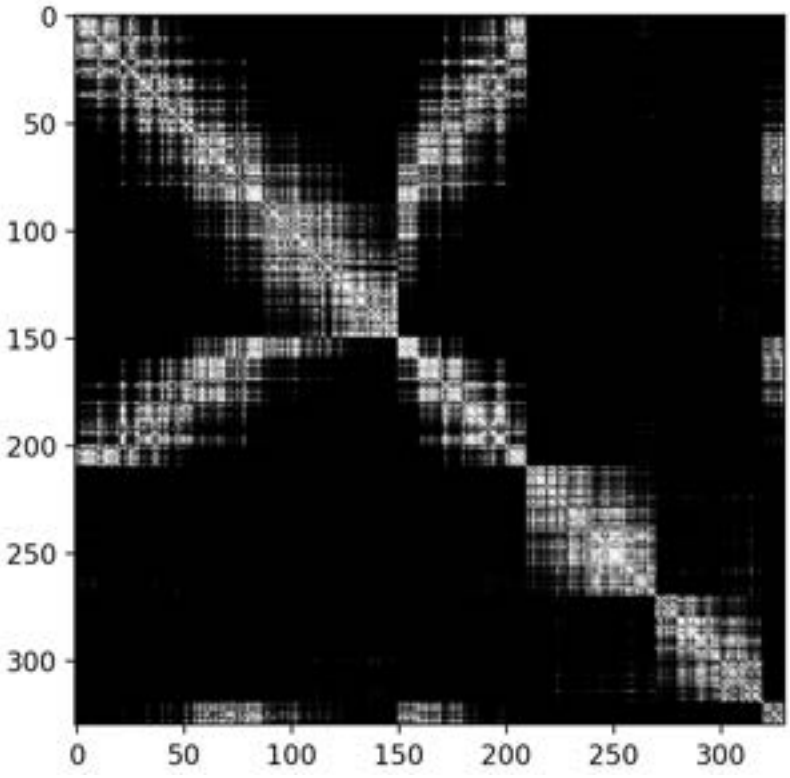

Figure 26 : Matrice d'affinités associée à l'exemple de spectral-clustering ci-dessus.

Nous testons ensuite l'influence de l'hyper-paramètre sigma sur un autre cas de répartition complexe de points (K=2) :

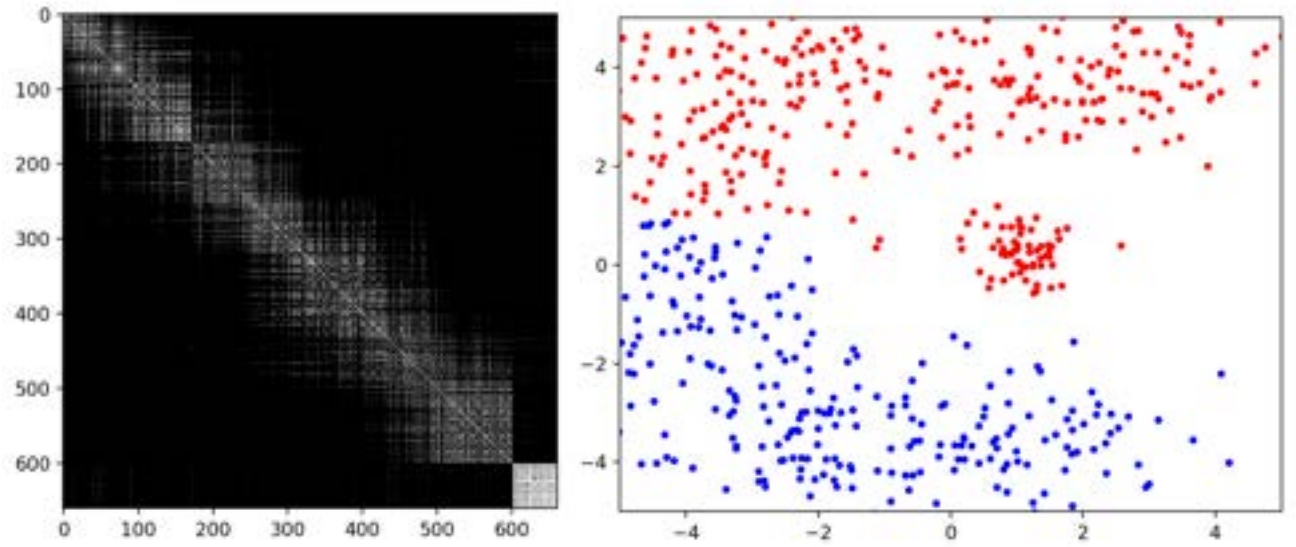

Figure 27 : Matrice d'affinité et représentation graphique du résultat pour sigma = 1.

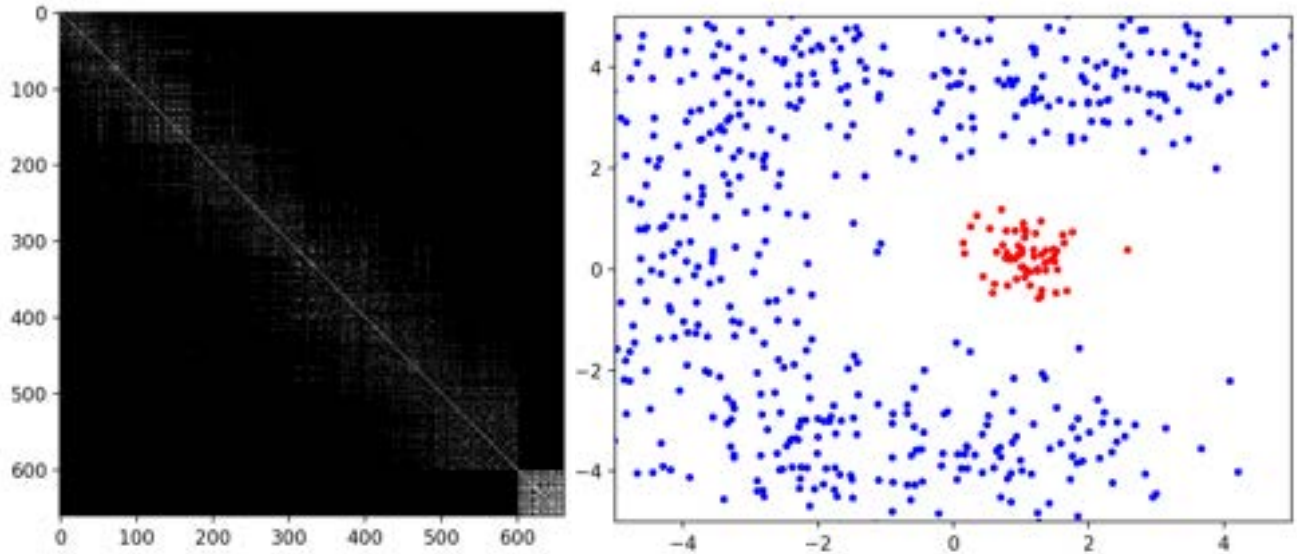

Figure 28 : Matrice d'affinité et représentation graphique du résultat pour sigma = 0.5.

Il semblerait que nous obtenons un résultat plus précis et plus déterministe pour un sigma faible, pour un sigma élevé les résultats sont d'avantages similaires à ce que pourrait renvoyer la méthode des K-means. On remarque également que plus sigma est faible, plus le temps de calcul devient long.

3. …

II - Application à la segmentation d'images

...