KDE

January 20, 2022

1 Introduction

Une des étapes fondamentales dans un projet d'apprentissage statistique est de comprendre les données à notre disposition. Ainsi, on peut s'interesser en particulier aux fonctions densités des variables. En effet, connaître la fonction densité qui génère une variable permet d'évaluer si ses valeurs extrêmes sont abérantes ou non. De plus, certains modèles font des hypothèses sur les distributions de probabilité des variables d'entrée. Respecter ces hypothèses nous permet de se rapprocher du cadre théorique développé autour du dit modèle.

Il existe trois types de variables: - qualitatives. - quantitatives discrètes. - quantitatives continus.

La repartition d'une variable qualitative peut être représentée par un graphique à bar tandis qu'on utilise un histogramme pour représenté la répartition des variables quantitatives.

Le tableau suivant présente les différences entre un histogramme et un graphique à barres verticales.

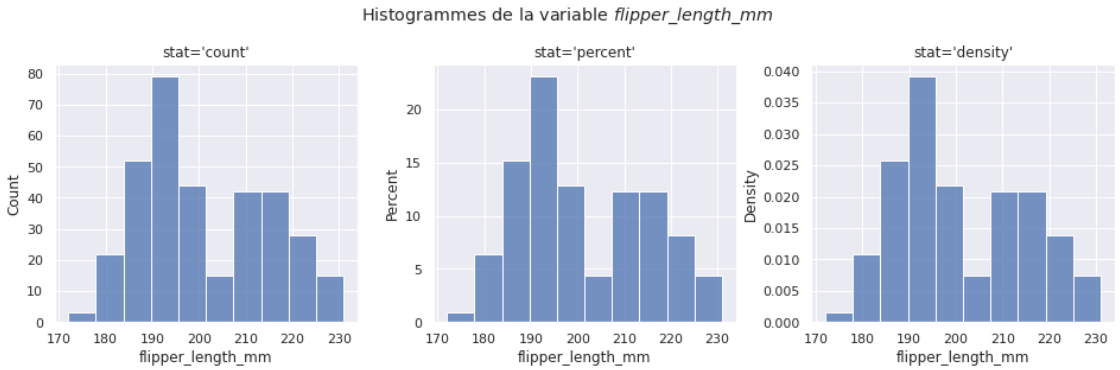
Concernant les variables quantitatives continues, on les discrétise pour construire l'histogramme.

La première partie traitera des avantages et inconvénients cette méthode. Puis, nous présenterons une méthode d'estimation de la fonction densité basée sur la notion de noyau appelée KDE pour Kernel Density Estimation. Enfin, nous présenterons des applications de cette méthode.

2 Histogramme

L'histogramme est un outil fréquemment utilisé pour montrer les caractéristiques principales de la distribution des données quantitatives discrètes ou continues.

Un histogramme sépare les valeurs possibles des données en classes ou groupes. Pour chaque groupe, on construit un rectangle dont la base correspond aux valeurs de ce groupe et la hauteur correspond à une fonction d'aggréégation sur le nombre d'observations dans le groupe tel que le comptage simple, le pourcentage ou la densité. L'histogramme a une apparence semblable au graphique à barres verticales, mais il n'y a pas d'écart entre les barres. En règle générale, l'histogramme possède des barres de largeurs égales.



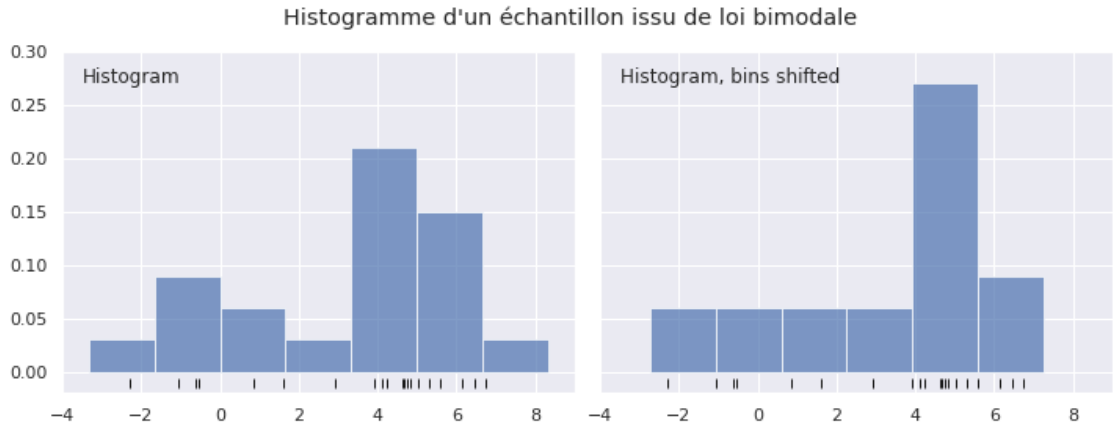
On choisit en général l'aggrégation par densité car cela permet de comparer la répartition de variables avec des variances différentes.

L'histogramme peut néamoins représenter de manière très différentes une même répartion en fonction du groupement des observations. Imageons cela par un exemple. On génère dans un premier temps un n-échantillon de variable parente X telle que :

$$X = pY + (1-p)Z$$

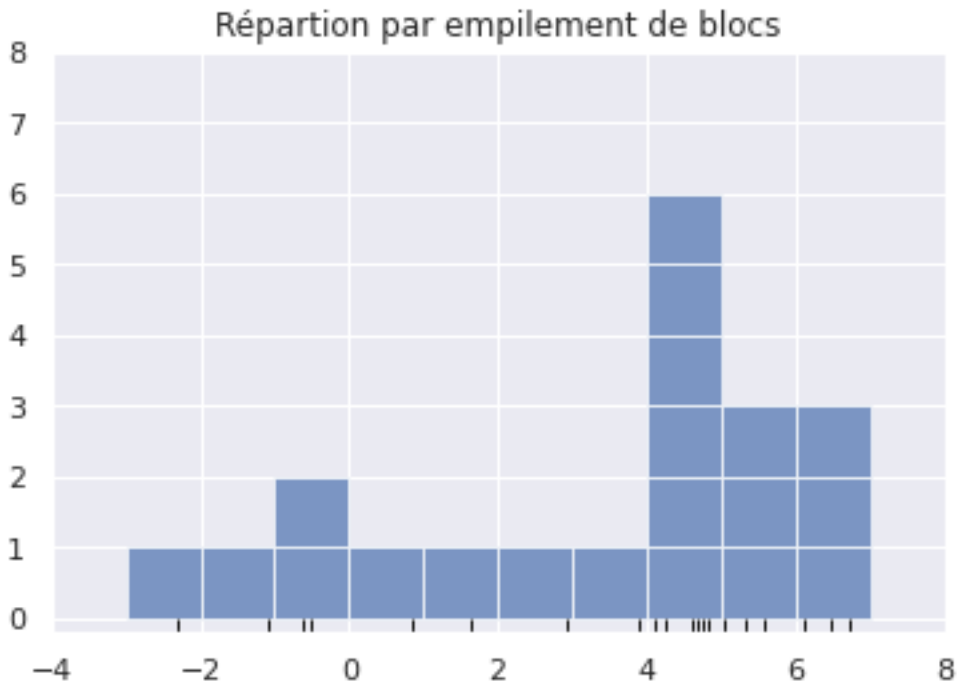
avec $p \sim \mathcal{B}(0.3)$, $Y \sim \mathcal{N}(0,\, 1)$ et $Z \sim \mathcal{N}(5,\, 1)$.

Puis dans un second temps, on représente notre échantillon ($n = 20$) avec des histogrammes sur la figure suivante.

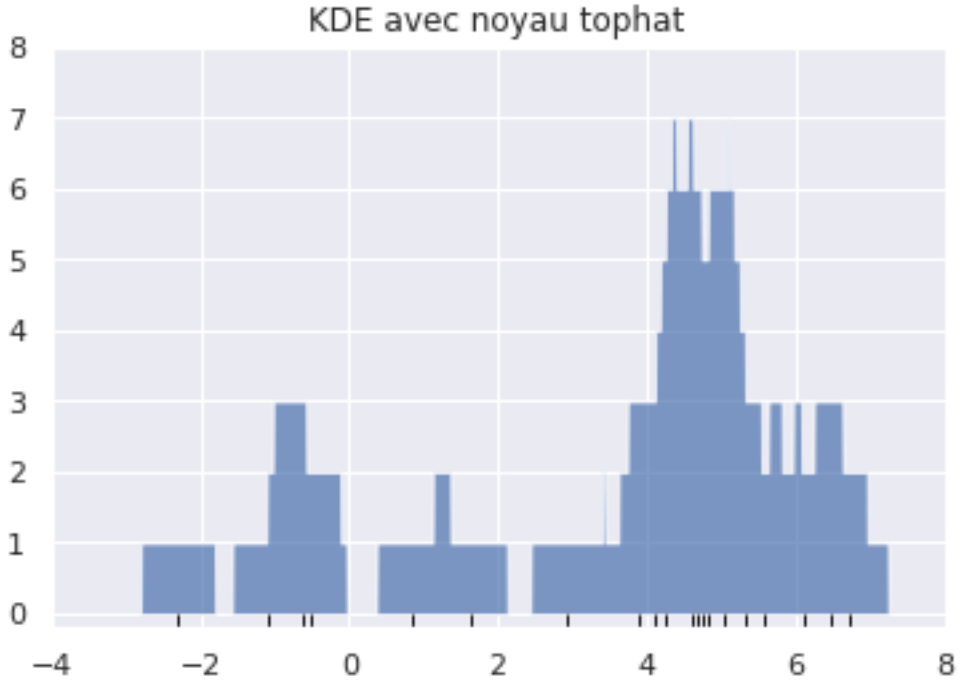


La figure de gauche semble effectivement représenter une distribution bimodale de deux lois gaussienne. Mais lorsque on translate les bornes des groupes légèrement vers la droite on obtient le graphique de droite qui semble représenté plutôt une distribution unimodale. Détaillons cela.

L'histogramme peut être vu comme un empilement de bloc. Plus il y a d'observations dans un intevalle plus il y a des blocs empilés.

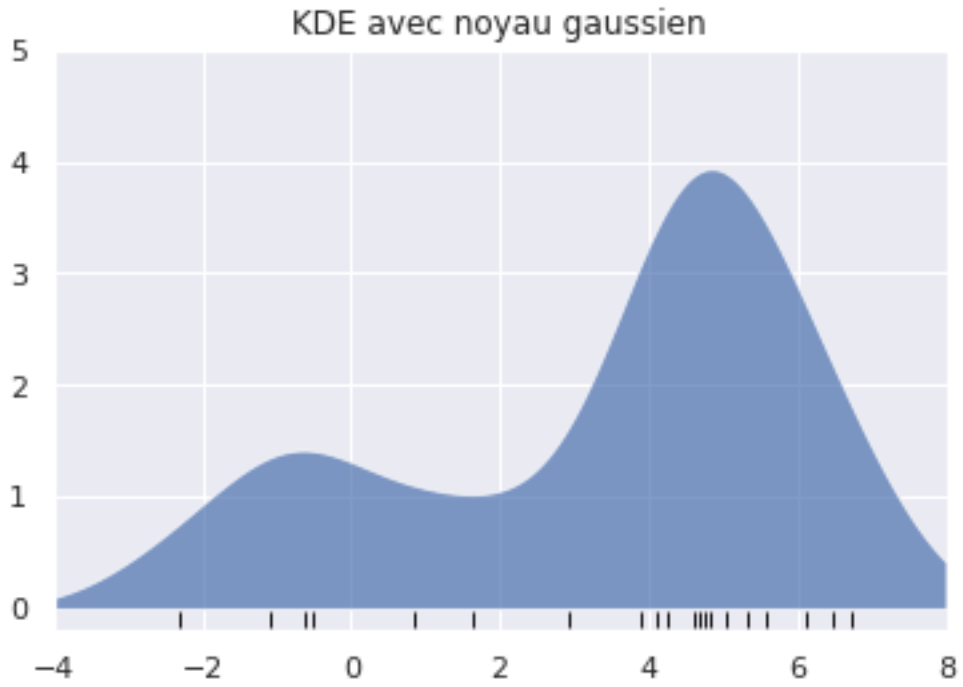


Le problème avec ces représentations est qu'à un point d'abcisse donné, la représentation peut être modifiée en fonction des groupements (cf. paramètre `bins`). Pour résoudre ce problème, on peut imaginer des blocs non pas positionnés sur des intervalles donnés mais centrés sur les observations. Cela donne le graphique suivant :



Cette représentation est plus robuste et également plus représentative de la répartition de notre échantillon.

Cependant, il est possible d'améliorer encore cette représentation. En effet, supposons qu'un bloc soit un carré de côté $2h$. Lorsqu'on somme les blocs comme dans la figure ci-dessus, on considère une densité uniforme sur l'intervalle $[x_i - h, x_i + h]$. Néanmoins, il est naturel de penser que les nouvelles observations ont de plus fortes probabilités d'apparaitre proche des observations actuelles plutôt qu'éloignées de ces dernières. Il est alors intéressant de remplacer les blocs par des fonctions plus lisses telles que des gaussiennes.



On obtient une représentation plus robuste avec moins de variance : les représentations varient peu entre différents n-échantillons.

Les deux graphiques ci-dessus ont été obtenu par une méthode appellé KDE expliquée plus en détails dans la partie suivante.

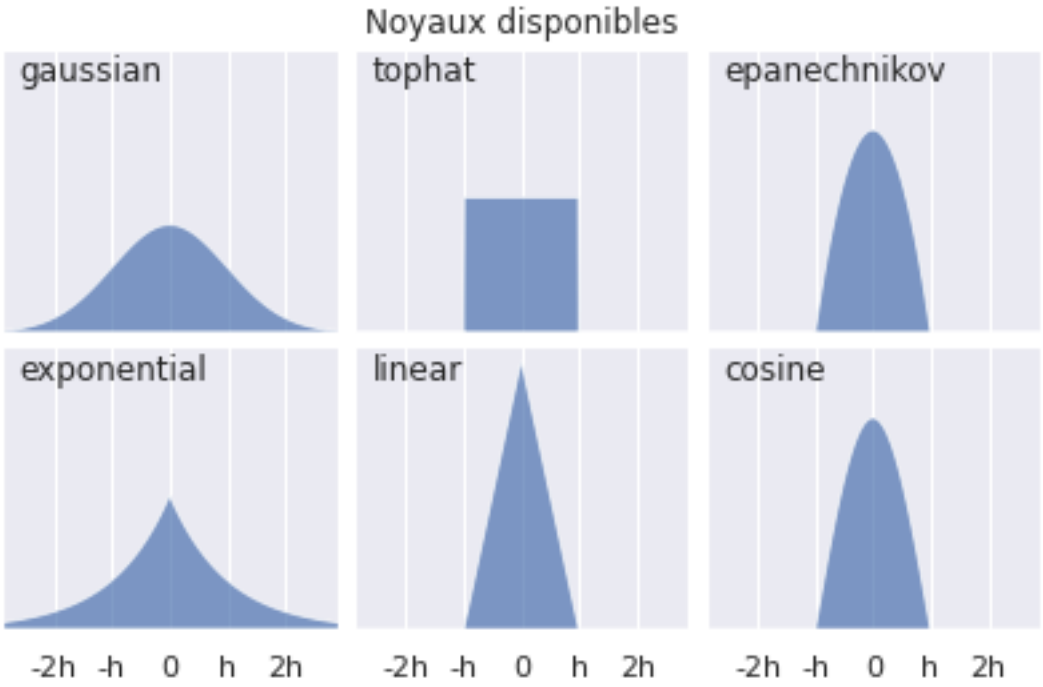
3 Kernel Density Estimation

Soit $x_1, x_2, ..., x_N \sim f$ un échantillon i.i.d. d'une variable aléatoire, ou plus généralement un groupe de points alors l'estimateur non-paramétrique de la densité par la méthode du noyau (=KDE) au point y est :

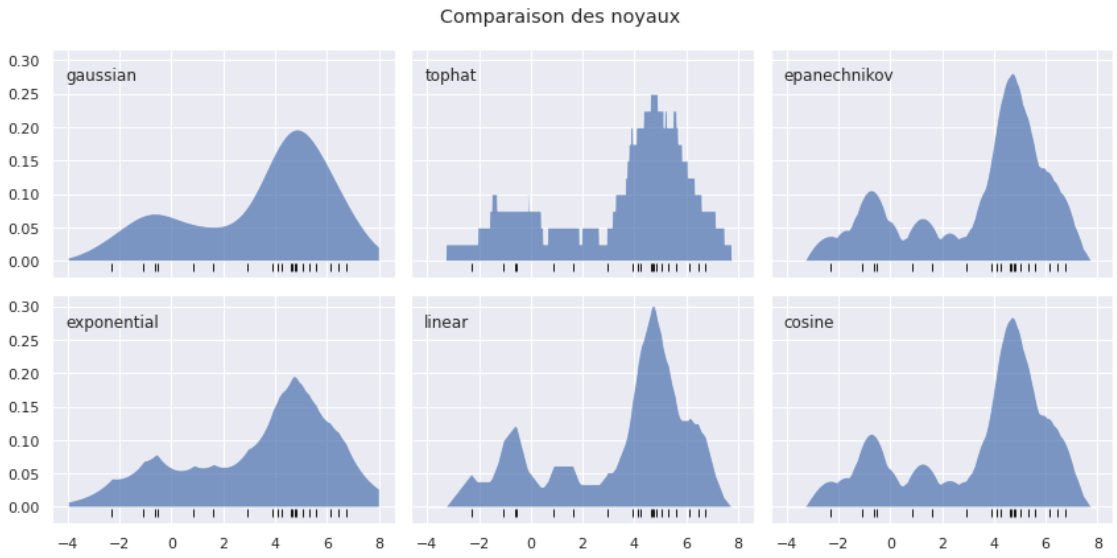
$$\widehat{f}_K(y) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(y - x_i; h\right)$$

où K est un noyau (kernel en anglais) et h un paramètre nommé fenêtre, qui régit le degré de lissage de l'estimation.

L'estimateur sklearn.neighbors.KernelDensity est une implémentation en Python de cette méthode. Le choix des N points utilisés est opéré par l'algorithme Ball Tree ou KD Tree et 6 noyaux sont proposés :

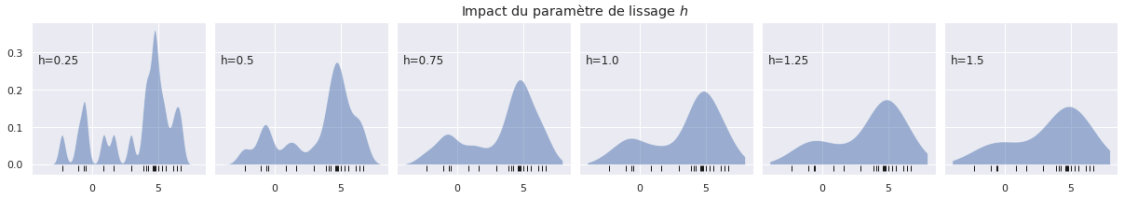


Appliquons sur notre jeu de donnée cet estimateur avec ces différents noyaux. On fixe $h = 1$.



On remarque que les noyaux `epanechnikov`, `linear`, et `cosine` donnent des résultats similaires avec un léger sur-ajustement. En effet, on pourrait penser que trois gaussiennes composent la fonction densité. Le noyaux `gaussian` est celui qui donne la répartition la plus lisse avec un bon ajustement. Le noyau `exponential` semble quand à lui sous-ajuster les données. Il est difficile de conclure avec le noyau `tophat` car le répartition n'est pas lissée.

Regardons à présent l'effet du paramètre h sur les estimations :



On observe donc que le paramètre h peut davantage modifier le résultat de l'estimation que le noyaux. Une valeur de h faible permettera d'avoir une estimation avec un faible biais mais avec une grande variace (risque d'overfitting). De manière symétrique, une valeur de h grande donnera un estimateur avec une faible variance mais avec un biais plus élévé (risque d'underfitting).

Comment choisir la valeur du paramètre h ?

Il existe des résultats théoriques donnant la valeur optimale h^*. Par exemple, dans la fonction `gaussian_kde` du module `scypi.stat`, la règle de Scott est utilisée. Ainsi, $h^* = n^{\frac{-1}{d+4}}$ en dimension d en particulier $h^* = n^{-\frac{1}{5}}$ en dimension 1.

Par ailleurs, à l'instar des modèles de machine learning classiques, il est possible de sélectionner ce paramètre par validation croisée. La fonction de perte à minimiser est la log-vraissemblance.

```
h* = 0.549 avec la règle de Scott
h* = 1.111 par validation croisée
```

On obtient une bien meilleure estimation de la densité avec h choisit par validation croisée lorsque qu'on met en relation les graphiques obtenus précédemment avec la loi parente.

4 Applications

À présent, nous allons étudier 2 applications de la méthode KDE.

4.1 Rééchantillonnage

Pour cette première application, nous allons utiliser la célèbre base de donnée de nombres manuscrits MNIST. L'objectif est de générer un nouvel échantillon à partir de cette dernière. Chaque image est représentée par un vecteur de taille 64.

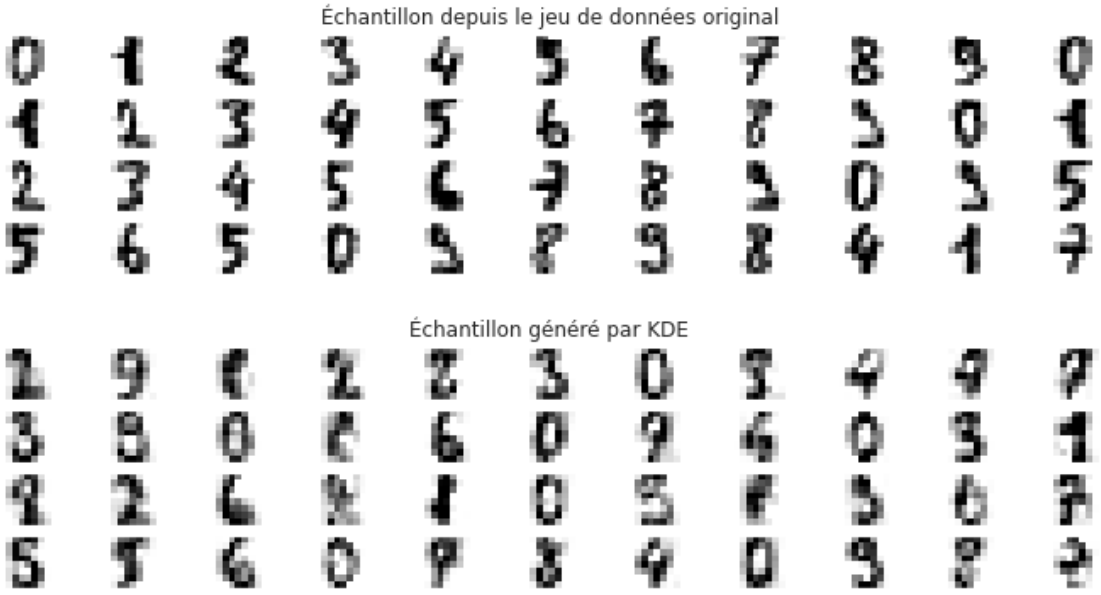
Premièrement, nous allons appliquer une ACP afin de palier aux problèmes liés à la grande dimension. On passe alors d'un espace de dimension 64 à un espace à 15 dimensions. Puis, sur ces composantes principales nous entrainons par validation croisée l'estimateur `KernelDensity` afin de

choisir la meilleure valeur pour notre paramètre de lissage h. Comme dit précedamment, le score utilisé est la log-vraissemblance.

Finalement, nous utilisons la méthode `sample` de notre estimateur pour généré les nouvels indivus (dans $\mathbb{R}^{15}$) avant d'appliquer la PCA inverse.

En passant nos vecteurs de taille 64 à des matrices de taille 8*8, on obtient le résultat ci-dessous.

```
best bandwidth: 3.79269019073225
```



Mais en détails, comment l'estimateur génère les nouvaux individus ?

Si on souhaite généré un nouvel individus, on tire au sort un point $x_i \in \mathbb{R}^l$ dans notre ensemble de points initial. Puis on génère aléatoirement un point selon une loi multinomiale centré sur x_i est de variance hI_p. Pour obtenir plusieurs individus, on itère le processus autant de fois que nécéssaire.

4.2 Classification supervisée

Pour la classification bayésienne naïve gaussienne, le modèle génératif est constitué de simples gaussiennes alignées sur les axes. Avec un algorithme d'estimation de densité comme KDE, nous pouvons supprimer l'élément "naïf" et effectuer la même classification avec un modèle génératif plus sophistiqué pour chaque classe. Il s'agit toujours d'une classification bayésienne, mais elle n'est plus naïve.

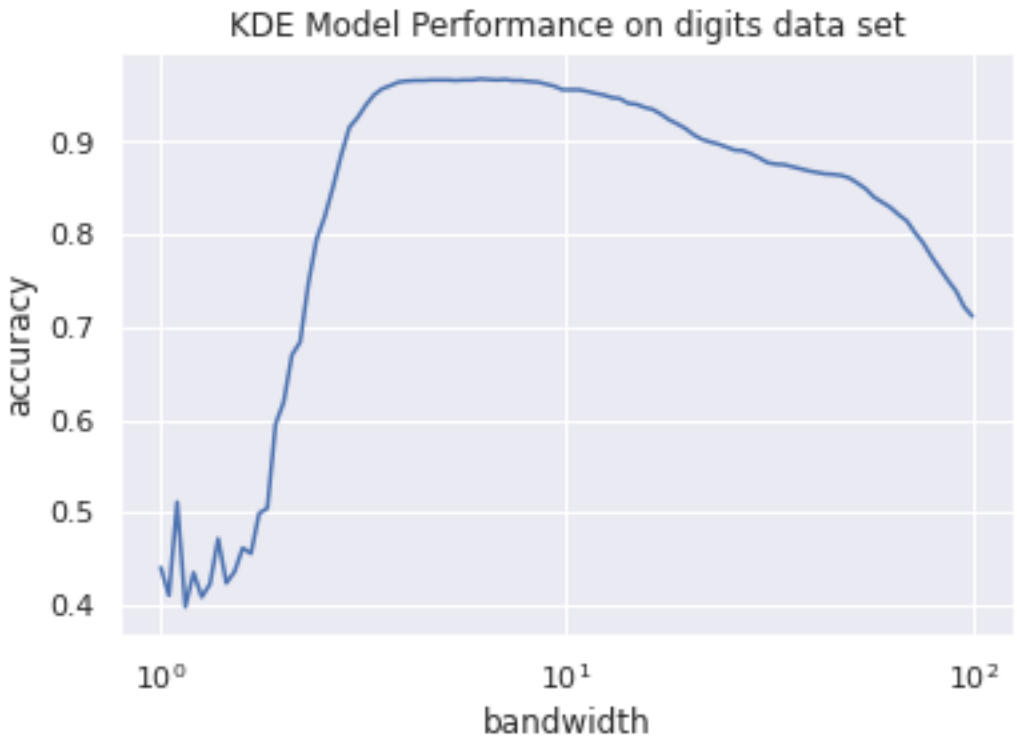
L'approche générale de la classification basé sur la méthode KDE est la suivante : - Divisez les données en groupe d'apprentissage selon les étiquettes. - Pour chaque ensemble, ajustez un estimateur KDE. Cela nous permet, pour toute observation x et étiquette y, de calculer une vraisemblance $\hat{\mathbb{P}}(x|y)$. - À partir du nombre d'exemples de chaque classe dans l'ensemble d'apprentissage, on calcule la probabilité apriori $\hat{\mathbb{P}}(y)$. - Pour un point inconnu x, la probabilité de chaque classe est $\mathbb{P}(y|x)$ $\hat{\mathbb{P}}(x|y)\hat{\mathbb{P}}(y)$. La classe qui maximise cette quantité est l'étiquette attribuée au point.

Voici le code qui implémente l'algorithme dans le cadre de Scikit-Learn :

```
class KDEClassifier(BaseEstimator, ClassifierMixin):
    """Bayesian generative classification based on KDE.

    Parameters
    ----------
    bandwidth : float
        the kernel bandwidth within each class
    kernel : str
        the kernel name, passed to KernelDensity
    """
    def __init__(self, bandwidth=1.0, kernel='gaussian'):
        self.bandwidth = bandwidth
        self.kernel = kernel

    def fit(self, X, y):
        self.classes_ = np.sort(np.unique(y))
        training_sets = [X[y == yi] for yi in self.classes_]
        self.models_ = [KernelDensity(bandwidth=self.bandwidth,
                                      kernel=self.kernel).fit(Xi)
                        for Xi in training_sets]
        self.logpriors_ = [np.log(Xi.shape[0] / X.shape[0]) for Xi in training_sets]
        return self

    def predict_proba(self, X):
        logprobs = np.array([model.score_samples(X)
                             for model in self.models_]).T
        result = np.exp(logprobs + self.logpriors_)
        return result / result.sum(1, keepdims=True)

    def predict(self, X):
        return self.classes_[np.argmax(self.predict_proba(X), 1)]
```

Appliquons notre l'algorithme de classification sur le jeu de données de chiffres manuscrits utilisé précédamment. Le meilleur paramètre de lissage h est encore choisit par validation croisée.

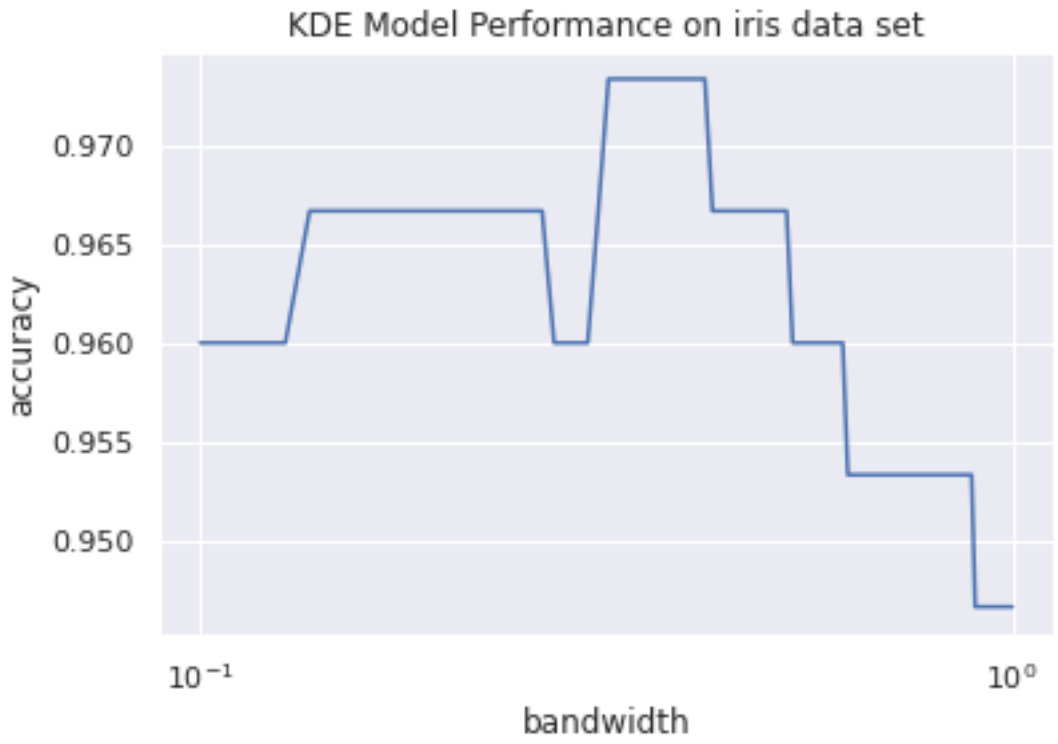
```
Best bandwidth = 6.136
KDEClassifier accuracy = 0.968
GaussianNB accuracy = 0.807
```




```
Output :
Best bandwidth = 6.136
KDEClassifier accuracy = 0.968
GaussianNB accuracy = 0.807
```

Testons à présent sur le data set Iris.

```
Best bandwidth = 0.318
KDEClassifier accuracy = 0.973
GaussianNB accuracy = 0.953
```




```
Output:
Best bandwidth = 0.318
KDEClassifier accuracy = 0.973
GaussianNB accuracy = 0.953
```

5 Conclusion

Nous avons étudié dans ce projet la méthode non paramétrique KDE. On l'utilise comunément pour améliorer la représention de la distribution d'un échantillon. Néanmoins, cette méthode obtient également des résultats visuellement satisfaisants dans le domaine du rééchantillonnage. De plus, nous avons construit d'un algorithme de classification basé sur cette méthode surpassant les performances du prédicteur Naïf de Bayes gaussien.

6 Références

1. Python Data Science Handbook, Jake VanderPlas
2. Guide utilisateur de l'algorithme KDE
3. Guide utilisateur de l'algorithme de recherche des K-ppv
4. Code source de l'algorithme KDE