

Devoir n°2

Pierre Snell

October 17, 2018

Sommaire

1	Estimation de densité par noyau	1
1.a	Estimation par histogramme	1
1.b	Estimation avec noyau boxcar	2
1.c	Comparaison noyeau vs K-nn	2
1.d	Comparaison pour le classement	3
2	Discriminants linéaires et k-plus proches voisins	3
2.a	Développement mathématique	3
2.b	Implémentation	4
2.c	Résultats	4
2.c.i	Avec 2 classes	4
2.c.ii	Avec 3 classes	5
2.d	Comparaison avec des modules existants	6
2.e	Knn	6

Liste des figures

1	Estimation par histogramme	2
2	Estimation par noyau tophat	2
3	Frontières de décision selon 2 classes	4
4	Frontières de décision selon 3 classes	5
5	Score des Knn en fonction du nombre de voisins pris en compte	6

Liste des tableaux

1	Comparaison des résultats avec des modules de scikitlearn	6
---	---	---

1 Estimation de densité par noyau

1.a Estimation par histogramme

On peut voir sur la figure 1 la vraie fonction de densité en rouge et les estimés par histogramme en bleu.

On remarque qu'au plus il y a d'échantillons par bins au plus l'approximation est fidèle. En effet, à défaut d'avoir des compartiments de plus en plus petits on obtient $\lim_{n \rightarrow \infty} \frac{\text{Nb echantillons}}{\text{compartiment}} = \infty$. Ce qui implique que $\hat{p}(x) \rightarrow p(x)$

Pour générer les échantillons, on crée un jeu de données entre -5 et 10 avec un pas faible, puis on utilise la fonction `numpy.random.choice()` pour nous permettre de tirer aléatoirement des échantillons en fonction de la distribution fournie dans l'énoncé. Enfin on effectue l'histogramme de ces données.

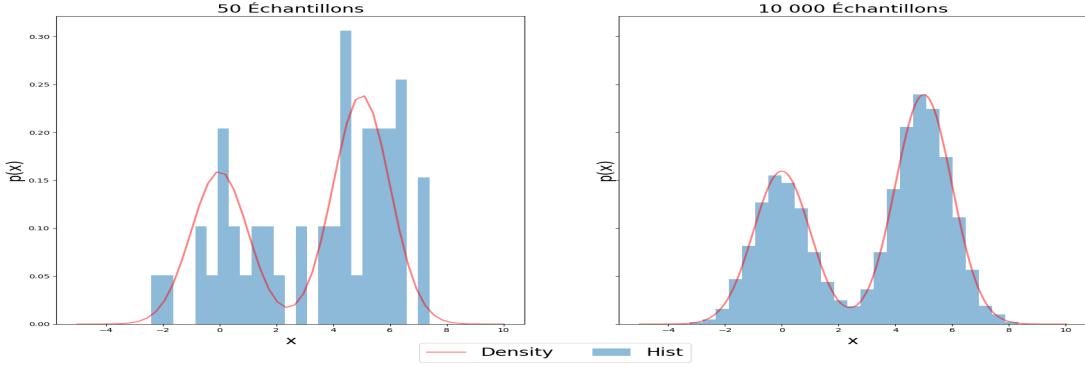


Figure 1: Estimation par histogramme

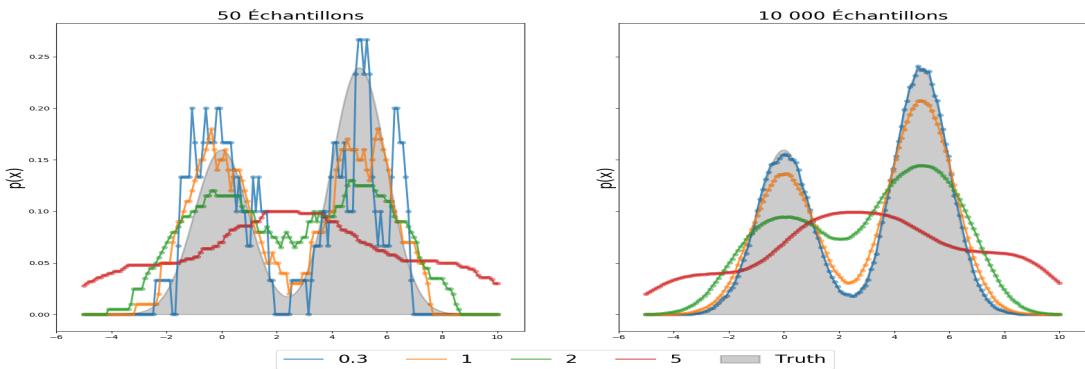


Figure 2: Estimation par noyau tophat

1.b Estimation avec noyau boxcar

On peut voir sur la figure 2 la vraie fonction de densité en gris plein et les estimées en fonction de leur taille de noyaux en couleur. Il est tout à fait normal d'obtenir des résultats moins précis avec seulement 50 données comparées à 10 000. Au plus du noyau est étroit au plus les données proches auront d'importance.

De ce fait, la courbe rouge qui correspond à la largeur la plus importante lisse beaucoup plus les données et fait perdre trop d'information.

De même, la verte (noyau plus étroit) suit mieux la courbe car elle donne plus d'importance aux données locales mais perd l'information du creux central.

On peut donc penser que pour une variation "rapide" comme ici, le meilleur pour estimer la courbe originelle semble donc être le noyau le plus étroit.

Cependant, un noyau plus étroit affecterait trop d'importance à chaque donnée et créerait une courbe trop dentelée. Pour d'autres applications en revanche, des noyaux plus larges peuvent être plus efficaces.

1.c Comparaison noyeau vs K-nn

Avec une méthode à noyau on peut changer la forme de celui-ci et donc son influence sur les données plus ou moins locales avec sa taille (largeur). En revanche la taille du noyau est un paramètre fixe qui reste le même tout au long de l'analyse. On compte simplement le nombre de données en convoluant le noyau sur le jeu de données. La répartition des données ne peut donc pas être modifiée et reste un paramètre intrinsèque. Ces méthodes sont donc bonnes pour analyser une répartition/densité

Le paradigme des K-plus proches voisins permet d'avoir une approche à noyau mais en pouvant faire varier la largeur du noyau centré sur une donnée. On ne regarde plus la répartition mais directement la distance de chaque point par rapport aux autres. Ensuite, en pondérant cette distance on peut calculer différentes informations mais en perdant la facilité d'analyse de la répartition des méthodes à noyau.

1.d Comparaison pour le classement

La méthode des K-ppv (K-nn) est bien plus simple et directe pour calculer une nouvelle valeur : il suffit de regarder les voisins d'un point inconnu et de lui affecter le label majoritaire de ceux-ci. De plus cette méthode est non supervisée. La méthode à noyau est bien plus complexe pour classifier, en revanche elle permet d'estimer des densités bien simplement que les Knn qui ne se préoccupent que de la distance entre les points et non de leurs répartitions. Les Knn, pour calculer une densité, se basent sur un ratio entre distance et nombre de plus proches voisins ce qui peut conduire à des singularités.

2 Discriminants linéaires et k-plus proches voisins

2.a Développement mathématique

Soit la fonction discriminante suivante :

$$h(\mathbf{x}|\mathbf{w}, w_0) = \mathbf{w}^\top \mathbf{x} + w_0 \quad (1)$$

avec \mathbf{w} la matrice des poids, et \mathbf{x} le vecteur d'entrée

et soit la fonction d'erreur :

$$E(\mathbf{w}, w_0 | \mathcal{X}) = \frac{1}{2} \sum_{\mathbf{x}^t \in \mathcal{Y}} \frac{[r^t - h(\mathbf{x}^t|\mathbf{w}, w_0)]^2}{\|\mathbf{x}^t\|^2} \quad (2)$$

avec \mathcal{Y} l'ensemble des données de \mathcal{X} (jeu entier) mal classées.

On à l'optimisation des poids suivante à chaque itération :

$$w_i = w_i + \Delta w_i \quad \forall i \in [0, D] \quad \text{D nombre de poids} = \#Features(X) + 1$$

Calculons Δw_i grâce à $E(\mathbf{w}, w_0 | \mathcal{X})$ de 2 :

$$\begin{aligned} \Delta w_i &= -\eta \frac{\partial E(\mathbf{w}, w_0 | \mathcal{X})}{\partial w_i} \\ &= -\eta \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{\mathbf{x}^t \in \mathcal{Y}} \frac{[r^t - h(\mathbf{x}^t|\mathbf{w}, w_0)]^2}{\|\mathbf{x}^t\|^2} = -\eta \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{\mathbf{x}^t \in \mathcal{Y}} \frac{[r^t - \mathbf{w}^\top \mathbf{x}^t - w_0]^2}{\|\mathbf{x}^t\|^2} \\ &= \frac{-\eta}{2} \sum_{\mathbf{x}^t \in \mathcal{Y}} \frac{\partial}{\partial w_i} \frac{r^{t2} + \mathbf{w}^\top \mathbf{x}^{t2} + w_0^2 - 2r^t \mathbf{w}^\top \mathbf{x}^t - 2r^t w_0 + 2\mathbf{w}^\top \mathbf{x}^t w_0}{\|\mathbf{x}^t\|^2} \\ &= \frac{-\eta}{2} \sum_{\mathbf{x}^t \in \mathcal{Y}} \frac{2\mathbf{w}^\top \mathbf{x}^{t2} - 2r^t \mathbf{x}^t + 2\mathbf{x}^t w_0}{\|\mathbf{x}^t\|^2} = \frac{-\eta}{2} \sum_{\mathbf{x}^t \in \mathcal{Y}} -2x_i^t \frac{r^t - (w_i x_i + w_0)}{\|\mathbf{x}^t\|^2} \end{aligned} \quad (3)$$

$$\Delta w_i = \eta \sum_{\mathbf{x}^t \in \mathcal{Y}} \mathbf{x}^t \frac{r^t - (\mathbf{w}^\top \mathbf{x}^t + w_0)}{\|\mathbf{x}^t\|^2}$$

De même en repartant de 3, mais avec $\frac{\partial}{\partial w_0}$:

$$\begin{aligned} \Delta w_0 &= \frac{-\eta}{2} \sum_{\mathbf{x}^t \in \mathcal{Y}} \frac{\partial}{\partial w_0} \frac{r^{t2} + \mathbf{w}^\top \mathbf{x}^{t2} + w_0^2 - 2r^t \mathbf{w}^\top \mathbf{x}^t - 2r^t w_0 + 2\mathbf{w}^\top \mathbf{x}^t w_0}{\|\mathbf{x}^t\|^2} \\ &= \frac{-\eta}{2} \sum_{\mathbf{x}^t \in \mathcal{Y}} \frac{2w_0 - 2r^t + 2\mathbf{w}^\top \mathbf{x}^t}{\|\mathbf{x}^t\|^2} = \frac{-\eta}{2} \sum_{\mathbf{x}^t \in \mathcal{Y}} -2 \frac{r^t - (w_i x_i + w_0)}{\|\mathbf{x}^t\|^2} \\ \Delta w_0 &= \eta \sum_{\mathbf{x}^t \in \mathcal{Y}} \frac{r^t - (\mathbf{w}^\top \mathbf{x}^t + w_0)}{\|\mathbf{x}^t\|^2} \end{aligned}$$

N.b : Avec $(g \circ f)' \quad g(x) = x^2 \quad f(x) = r^t - h(\mathbf{x}^t|\mathbf{w}, w_0)$ on obtient le même résultat pour Δw_i et Δw_0 en une ligne

2.b Implémentation

Le code commenté de l'implémentation se trouve [sur mon github](#) (ou remis sur monPortail). Il contient tout le docset et des indications sur les hyper-paramètres et la doc Pep-8 de chacune des classes et méthodes.

2.c Résultats

2.c.i Avec 2 classes

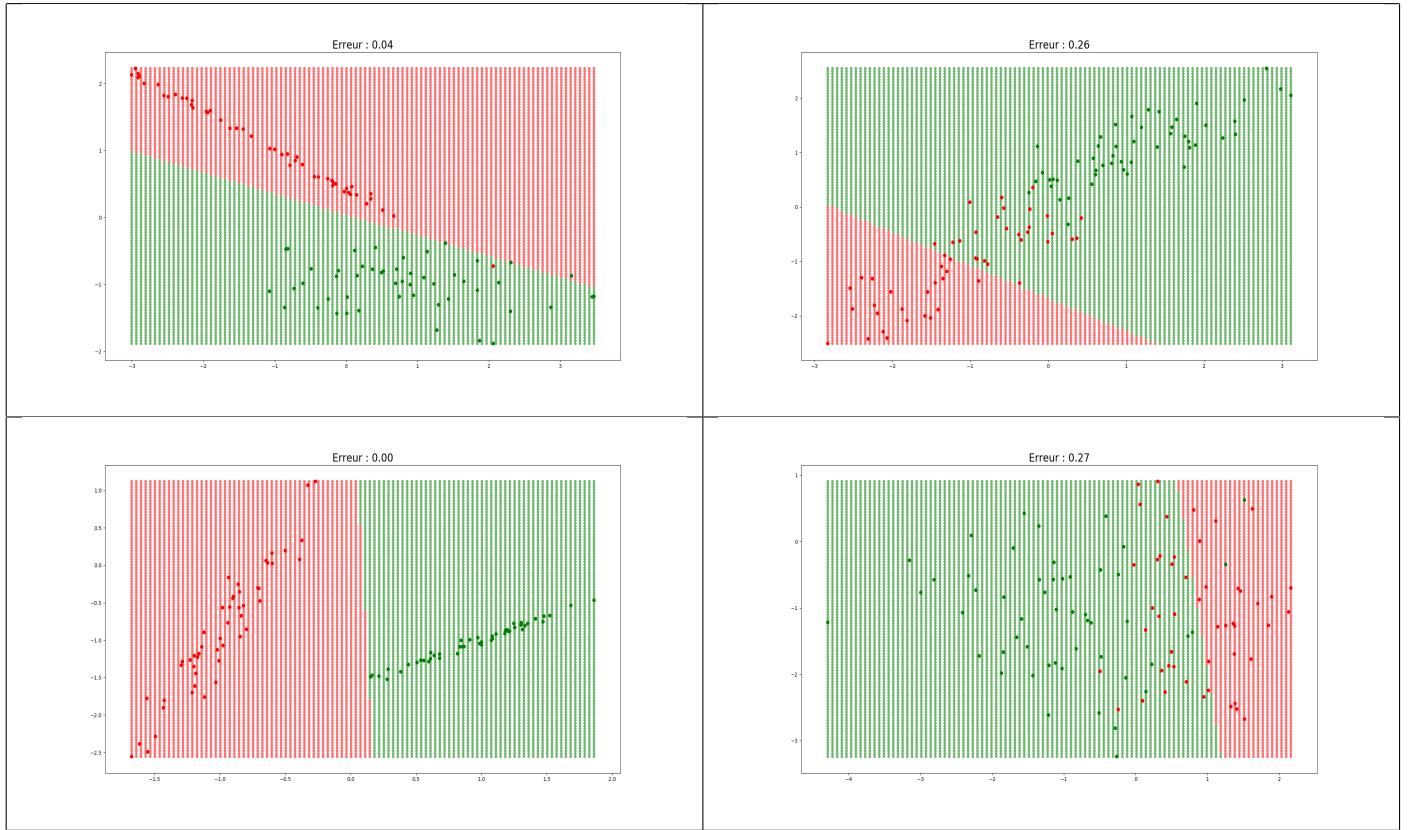


Figure 3: Frontières de décision selon 2 classes
Taux moyen d'erreur 10%-, (axes : Feature1 et Feature2 et classes verte 0 rouge 1 peu important)

On peut voir sur la figure 3 les différents types de classification que l'on peut obtenir. En moyenne le classificateur performe très bien (90%+) et parfaitement sur des données linéairement séparables. Sur la première sous-figure on peut voir une bonne classification, malgré un point rouge qui fait dévier la séparation. La classification reste quand même très bonne et la majorité des classements ressemble à cela.

La troisième sous-figure (bas gauche) montre une classification parfaite mais ne respectant pas les vastes marges (c'est l'inconvénient du modèle) qui mais n'empêche pas une classification optimale.

La sous-figure 2 (haut droit) montre un exemple de convergence dans un minimum local (tout de même assez rare). La frontière de décision n'est pas bonne mais le critère d'arrêt et la convergence des poids mène à ce type de résultats dans de certains cas.

Enfin, sous-figure 4, avec des données non linéairement séparables (cas extrême). La façon de descendre le gradient, de calculer les poids ne permet pas de classifier correctement un nuage de points comme celui-ci.

Le modèle implémenté classifie donc de manière générale très bien sur quasiment tous les jeux de données hormis certains cas très spécifiques. (Cf 1) Le seul reproche est celui lié à l'erreur de 2 qui prend l'erreur d'un point éloigné de façon trop importante et fait dévier les frontières de décisions.

2.c.ii Avec 3 classes

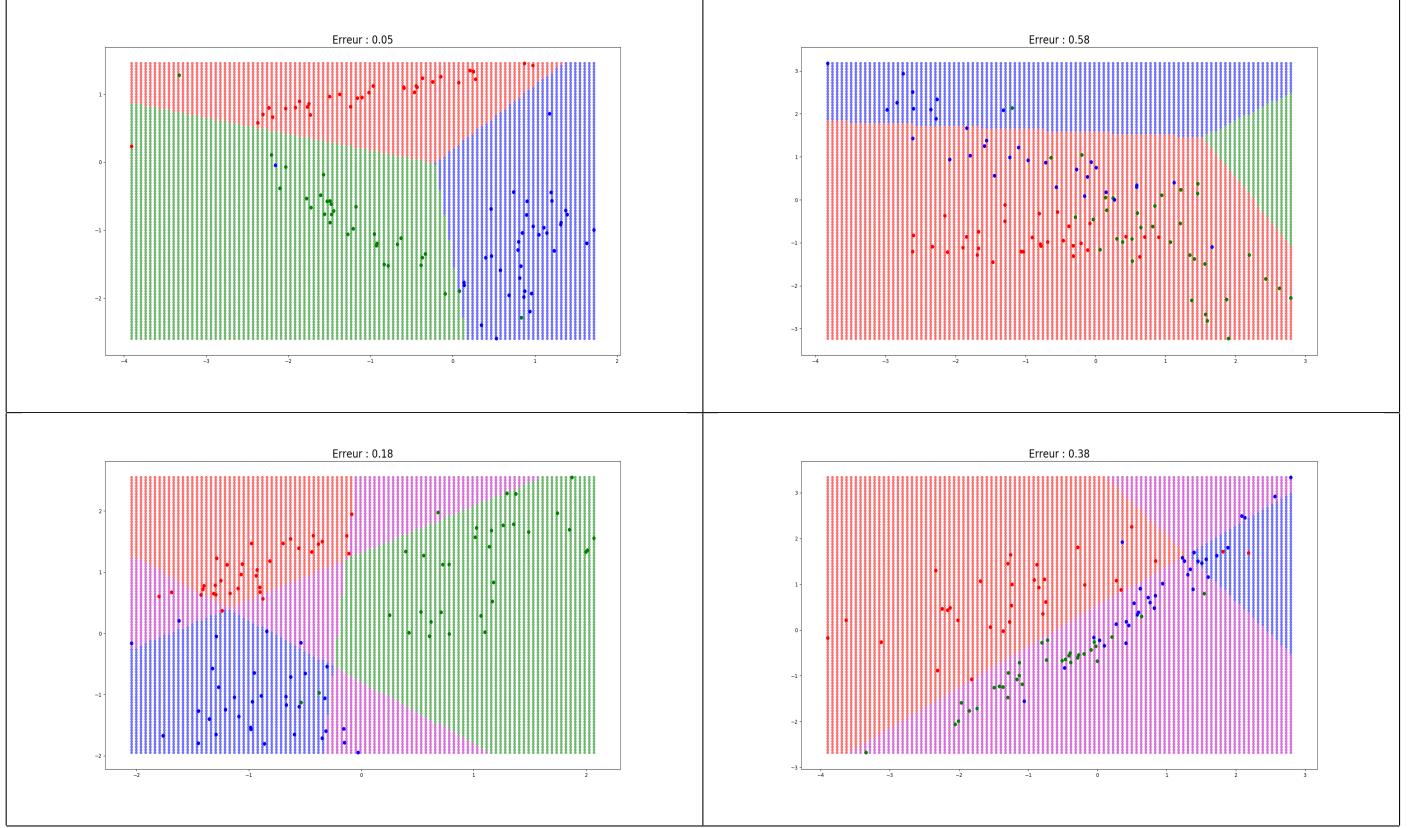


Figure 4: Frontières de décision selon 3 classes

Taux moyen d'erreur 10%, (axes : Feature1 et Feature2 et classes verte 0 rouge 1 bleu 2 rose ambigu peu importants)

Le classificateur à trois classes performe très bien malgré quelques erreurs sur certains jeux de données spéciaux. Il obtient en moyenne 90%+ de réussite.

Sur la sous-figure 1 de la figure 4 on peut voir le type de la majorité des classements avec la méthode argmax (voir 4).

Dans la sous-figure 2, on peut voir les erreurs qui arrivent quelquefois avec une mauvaise convergence (minimum local / critère d'arrêt atteint trop vite). Ce cas est plutôt rare mais plus utile pour la compréhension.

Les sous-figures du bas représentent le même algorithme mais avec un ajout d'une zone d'ambiguïté (voir 5). Celle en bas à gauche montre le cas général de classification. Celle à droite, un problème de convergence, ici sûrement lié aux données un peu trop proches.

Dans la majorité des cas l'algorithme classifie parfaitement les données avec un taux d'erreur très faible. (Cf 1)

Annexe :

de $h(\mathbf{x})$ de 1 on peut prédire les classes selon deux méthodes :

$$\text{Méthode de la valeur positive } h(x) = \begin{cases} C_i & \text{si } h_i(\mathbf{x}) \geq 0 \text{ } \& \text{ } h_j(\mathbf{x}) < 0 \text{ } \forall i \neq j \\ \text{ambiguïté} & \text{autrement} \end{cases} \quad (4)$$

$$\text{Méthode argmax : } h(\mathbf{x}) = \arg \max_{\substack{C_k \\ C_i=C_1}} h_i(\mathbf{x}) \quad (5)$$

2.d Comparaison avec des modules existants

Classifieur	Paramètres	Erreur Cancer Train	Erreur Cancer Test	Erreur Iris Train	Erreur Iris Test
Devoir	$\epsilon = 1^{-6}$ $\eta = 1^{-4}$	0.032	0.043	0.099	0.106
Linear :	Default	0.0351	0.042	0.016	0.020
Perceptron :	Default	0.047	0.060	0.243	0.23
Logistic :	Default	0.034	0.0421	0.163	0.18

Table 1: Comparaison des résultats avec des modules de scikitlearn

du 2.c.ii. selon la prédiction argmax 4 (valeur positive (5) donne quasiment les mêmes résultats).

Sur le jeu d'iris, X a 4 features et y compte deux classes, on utilise le discriminant linéaire de la section 2.c.i.

Sur le dataset du cancer, le modèle implémenté performe très bien. Il obtient les mêmes taux d'erreur que des modèles "professionnels".

En revanche, sur le jeu d'iris, la méthode du discriminant linéaire de scikitlearn performe avec un taux d'erreur environ 10 fois inférieur. L'hypothèse du solveur différent n'explique pas non plus cette différence (test effectué avec le solveur par décomposition en valeurs singulières (SVD) et avec le least mean square error comme notre modèle), l'implémentation de scikitlearn reste meilleure sur ce jeu de données.

Le modèle implémenté pour le devoir reste dans le taux d'erreur moyen des autres modèles ce qui confirme quand même son bon fonctionnement. De plus, une condition a été ajoutée pour s'assurer de la bonne convergence, maintenant, $\text{ErreurPrécédente} - \text{ErreurActuelle} < \epsilon$ ET $\text{ErreurActuelle} < \xi$ avec ξ arbitraire. La méthode des moments aurait aussi été possible mais une condition aussi simple était bien plus rapide à implémenter et donne de très bons résultats.

2.e Knn

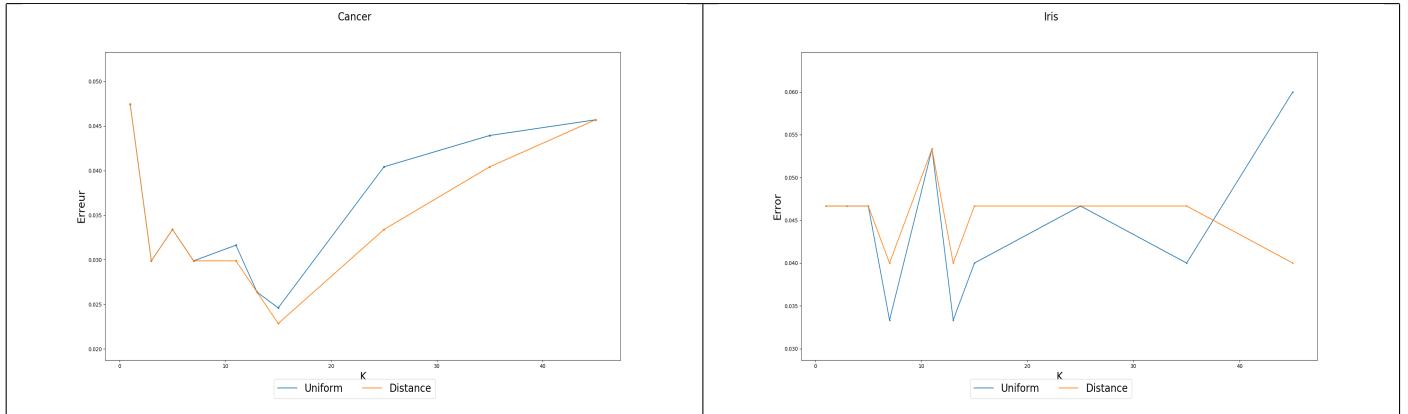


Figure 5: Score des Knn en fonction du nombre de voisins pris en compte

On remarque qu'à partir d'un certain nombre de voisins, l'erreur augmente drastiquement. En effet, en prenant trop de voisins en compte, les labels hors de la classe que nous voulons isoler (donc labels trop loin, ceux de l'autre classe) seront comptés, ce qui augmente l'erreur.

On peut aussi voir que trop peu de voisins ne suffisent pas à créer une bonne moyenne des labels environnants.

Un maximum pour K peut être trouvé par une analyse comme celle-ci ou par un grid search. Certains pics peuvent sûrement correspondre à des "îlots" de données de l'autre classe au sein d'un groupe de la classe que l'on souhaite prédire.

Les poids uniformes signifient que chaque voisin a autant d'importance, qu'il importe sa distance.

Les paramètres par défaut des trois autres classificateurs n'ont pas pu être trouvés dans la documentation car ils utilisent des solveurs qui n'ont pas les mêmes que celui implémenté dans le devoir. Il n'est donc pas possible de les comparer.

Sur le jeu du cancer, X a 30 features et y compte 2 classes. On utilise la méthode du un contre tous

Les poids distances signifie que chaque voisin à un poids pondéré par l'inverse de sa distance.
Concernant le jeu cancer, cela semble clair et intuitif que la pondération par distance soit la plus logique et la meilleure. Pour le jeu d'iris la solution ne semble pas évidente, la distance entre les données ne semble pas être un critère.

Pour les deux jeux de données l'erreur reste très faible en raison de la non-linéarité du modèle ce qui lui confère des performances bien meilleures que les modèles linéaires décrits dans toutes les parties précédentes.