COURS : ALGORITHMES POUR L'INTELLIGENCE ARTIFICIELLE
== ALGORITHME KNN ==

I) UN CLASSIFICATEUR OPTIMAL : LE CLASSIFICATEUR DE BAYES

Considérons deux densités conditionnelles $f_{\triangle}(x) = f_{X|Y=\triangle}(x)$ et $f_{O}(x) = f_{X|Y=O}(x)$ (dans notre exemple, chaque densité est un mélange de deux gaussiennes), décrivant les probabilités qu'un élément $x = (x_1, x_2)^T \in \mathbb{R}^2$ appartienne à la classe y = « triangle » ou « cercle » :

$$f_{\triangle}(x) = \sum_{m=1}^{2} \omega_{\triangle m}\, \mathcal{N}(x; \mu_{\triangle m}; \Sigma_{\triangle m}) \qquad f_{O}(x) = \sum_{m=1}^{2} \omega_{Om}\, \mathcal{N}(x; \mu_{Om}; \Sigma_{Om})$$

$$\mathcal{N}(x; \mu; \Sigma) = \frac{1}{(2\pi)\sqrt{det(\Sigma)}} exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

… avec $\omega_{\cdot m} \geq 0$ et $\sum_{\cdot m} \omega_{\cdot m} = 1$.
(voir https://informatique-f1.fr/jupyterlite/lab/index.html?path=Densites.ipynb pour les valeurs des paramètres des vecteurs moyennes $\mu_{\cdot m}$, des matrices de covariance $\Sigma_{\cdot m}$et des poids de mélange $\omega_{\cdot m}$ de cet exemple)

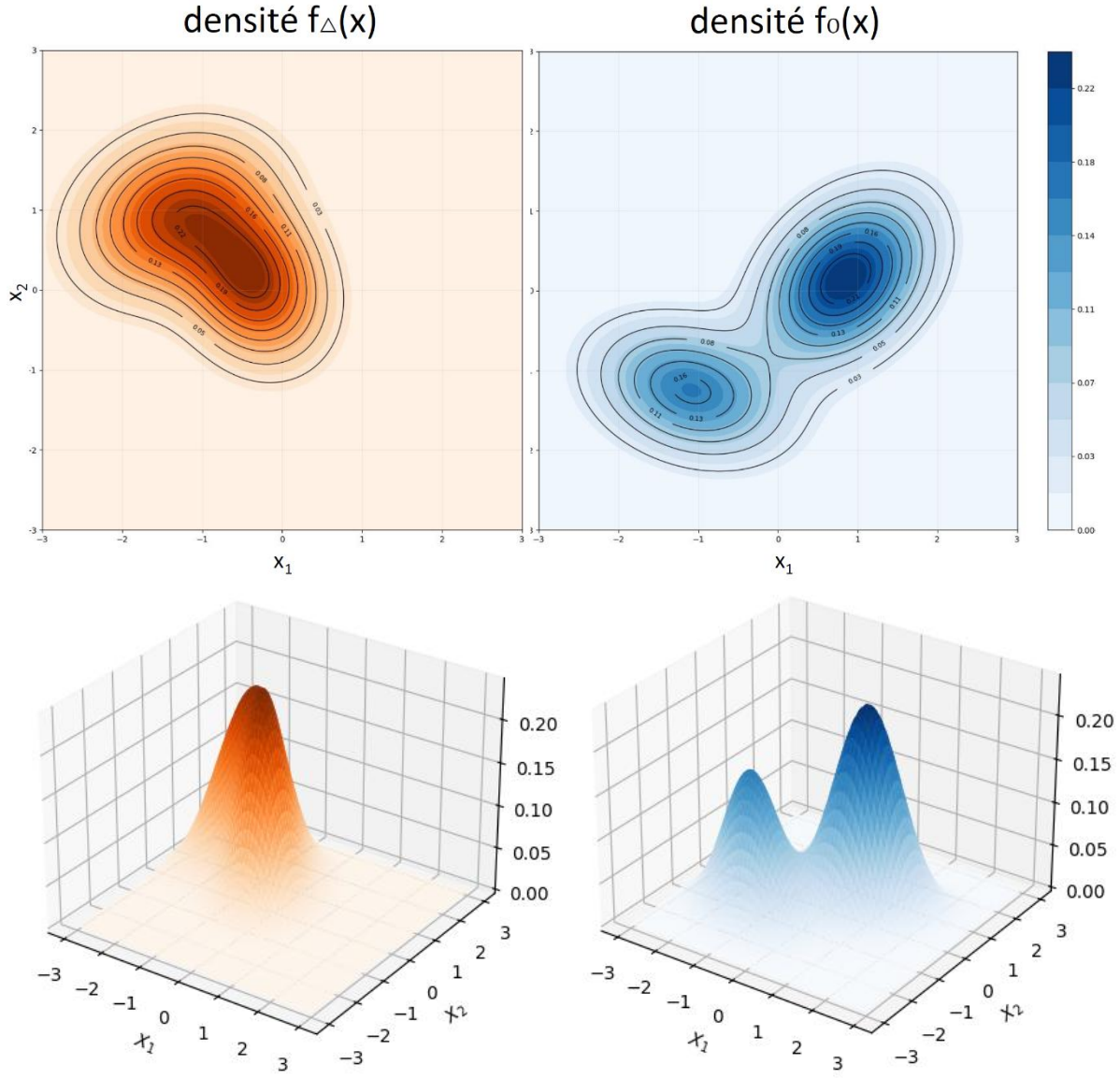

Figure 1 : Densité de probabilité $f_{\triangle}(x)$ et $f_{O}(x)$

On tire maintenant 80 triangles et 80 cercles en suivant les densités de probabilités $f_{\triangle}(x)$ et $f_{O}(x)$:

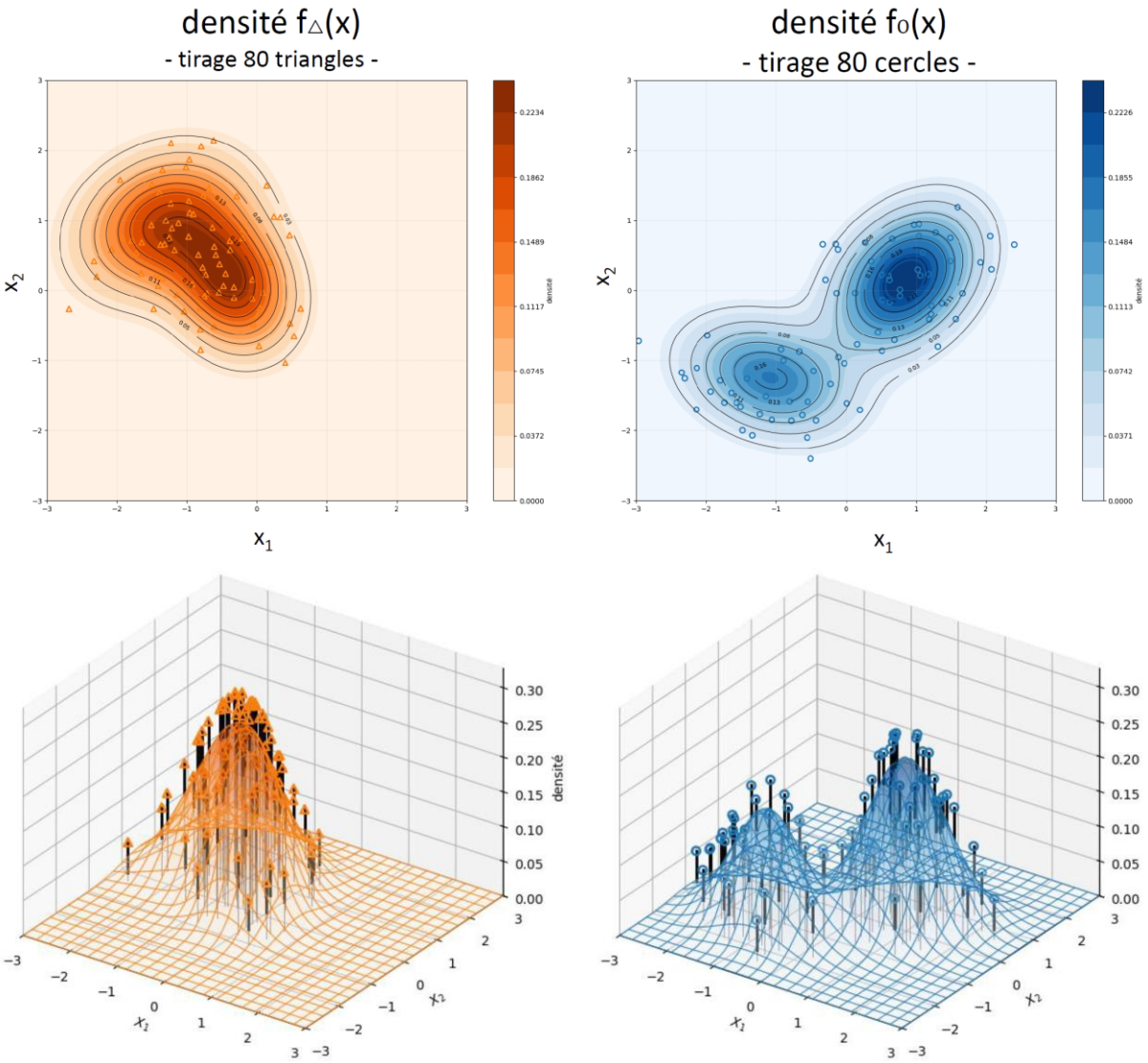

Figure 2 : Tirage de 80 triangles et 80 cercles selon les densités de probabilités $f_{\triangle}(x)$ et $f_{O}(x)$

Pour une observation de test $x_0 = ((x_0)_1, (x_0)_2)^T$, la probabilité de classification correcte est alors maximisée, en moyenne, par un classificateur très simple qui affecte chaque observation à la classe la plus probable, compte tenu des valeurs de ses caractéristiques $(x_0)_1$ et $(x_0)_2$. Autrement dit, il suffit d'affecter à une observation de test x_0, la classe j pour laquelle la probabilité conditionnelle :

$$Pr(Y = j \mid X = x_0)$$

… est la plus forte.

Ce classificateur très simple est appelé le *classificateur de Bayes*. Dans notre problème, où il n'y a que deux valeurs possibles pour la réponse, le classificateur de Bayes consiste à prédire la classe « triangle » si $Pr(Y = "triangle" \mid X = x_0) > 0.5$, et la classe « cercle » sinon.

Pour obtenir la probabilité conditionnelle $Pr(Y = \triangle \mid X = x_0)$ à partir des densités de probabilité $f_{\triangle}(x)$ et $f_{\mathrm{O}}(x)$, on applique la formule de Bayes :

$$P\big(Y = \triangle \mid X \in \mathcal{B}_{\varepsilon}(x_0)\big) = \frac{P(X \in \mathcal{B}_{\varepsilon}(x_0) \mid Y = \triangle) \cdot P(Y = \triangle)}{P\big(X \in \mathcal{B}_{\varepsilon}(x_0)\big)}$$

On définit : $P(Y = \triangle \mid X = x_0) := \lim_{\varepsilon \to 0} P\big(Y = \triangle \mid X \in \mathcal{B}_{\varepsilon}(x_0)\big)$

$$= \lim_{\varepsilon \to 0} \frac{P(X \in \mathcal{B}_{\varepsilon}(x_0) \mid Y = \triangle) \cdot P(Y = \triangle)}{P(X \in \mathcal{B}_{\varepsilon}(x_0) \mid Y = \triangle) \cdot P(Y = \triangle) + P(X \in \mathcal{B}_{\varepsilon}(x_0) \mid Y = \mathrm{O}) \cdot P(Y = \mathrm{O})}$$

Comme les densités conditionnelles $f_{\triangle}(x)$ et $f_{\mathrm{O}}(x)$ sont continues en x_0, on a :

$$P(X \in \mathcal{B}_{\varepsilon}(x_0) \mid Y = k) \sim f_{\mathrm{k}}(x_0) \cdot Vol\big(\mathcal{B}_{\varepsilon}(x_0)\big)$$

On obtient donc :

$$P(Y = \triangle \mid X = x_0) = \frac{f_{\triangle}(x_0) \cdot P(Y = \triangle)}{f_{\triangle}(x_0) \cdot P(Y = \triangle) + f_{\mathrm{O}}(x_0) \cdot P(Y = \mathrm{O})}$$

Dans notre cas, on considère les classes équiprobables : $\pi_{\triangle} = \pi_O = 0.5$. On obtient alors :

$$P(Y = \triangle \mid X = x_0) = \frac{f_{\triangle}(x_0)}{f_{\triangle}(x_0) + f_{\mathrm{O}}(x_0)}$$

Ainsi, prédire la classe « triangle » si $P(Y = \triangle \mid X = x_0) > 0.5$ revient géométriquement à tracer une limite sur la condition $f_{\triangle}(x_0) = f_{\mathrm{O}}(x_0)$, c'est-à-dire la ligne où les deux surfaces des densités se croisent.

La ligne violette en pointillés sur la Figure 3 représente cette limite. On l'appelle la *frontière de décision de Bayes*. La prédiction du classificateur de Bayes est déterminée par cette frontière de décision : une observation qui se situe du côté orange de la frontière sera affectée à la classe orange, et de même une observation située du côté bleu de la frontière sera affectée à la classe bleue.

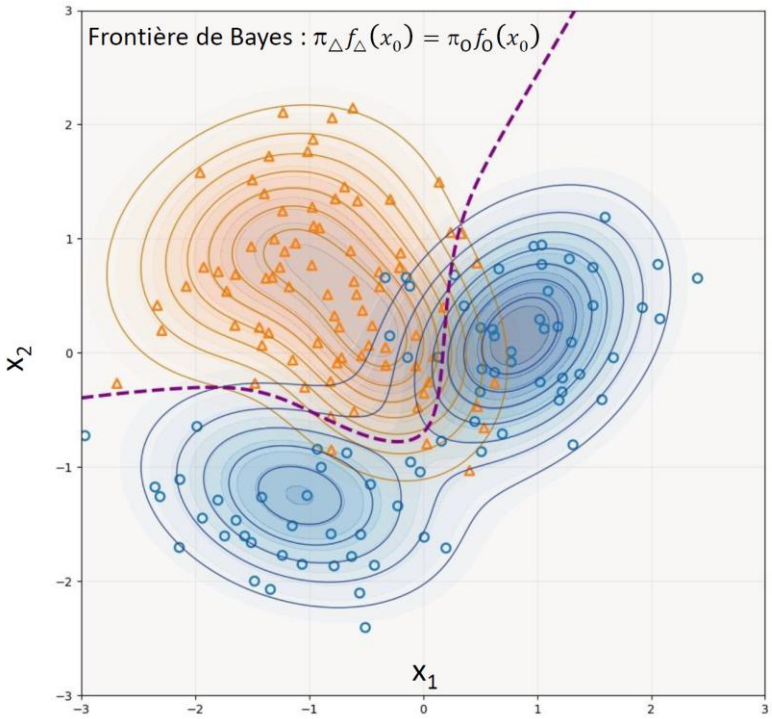

Figure 3 : Frontière de décision de Bayes

Le classificateur de Bayes produit le plus faible taux d'erreur de test possible, appelé le *taux d'erreur de Bayes*. Comme le classificateur de Bayes choisira toujours la classe pour laquelle la probabilité conditionnelle $Pr(Y = j \mid X = x_0)$ est la plus grande, le taux d'erreur en $X = x_0$ sera $1 - max_j\{Pr(Y = j \mid X = x_0)\}$. De façon générale, le taux d'erreur de Bayes global est donné par :

$$1 - \mathbb{E}\left[\max_j Pr(Y = j \mid X = x_0)\right]$$

… où l'espérance moyenne cette probabilité sur toutes les valeurs possibles de X. Pour nos données simulées, le taux d'erreur de Bayes est supérieur à zéro parce que les classes se chevauchent dans la population réelle pour certaines valeurs de x_0. Le taux d'erreur de Bayes est analogue à l'erreur irréductible, discutée au chapitre I.3.1.

En théorie, l'idéal serait de toujours prédire des réponses qualitatives à l'aide du classificateur de Bayes. Mais, avec des données réelles, nous ne connaissons pas les densités $f_{\mathrm{X|Y=j}}(x)$ ni les probabilités a priori $\pi_j = P(Y = j)$. Par conséquent, la probabilité a posteriori $Pr(Y = j \mid X = x_0)$ est inconnue.

Le classificateur de Bayes sert de référence idéale inatteignable, à laquelle on compare les autres méthodes. De nombreuses approches cherchent à estimer la distribution conditionnelle de Y sachant X, puis à classer une observation donnée dans la classe ayant la probabilité estimée la plus élevée. Parmi ces méthodes, la méthode des *k plus proches voisins* (*K-nearest neighbors, KNN*) que nous allons étudier maintenant cherche à établir localement la probabilité a posteriori à partir des voisins d'une observation.

II) LA MÉTHODE DES K PLUS PROCHES VOISINS (KNN)

II.1. Principe de fonctionnement

Étant donné un entier strictement positif k et une observation de test x_0, le classificateur KNN commence par identifier les k points de l'échantillon d'apprentissage les plus proches de x_0, notés $\mathcal{N}_0$. La notion de « proximité » dépend d'une distance d(·,·) choisie sur l'espace des variables explicatives : géométriquement, $\mathcal{N}_0$ correspond aux points situés dans une boule centrée en x_0 (au sens de la distance d), de rayon égal à la distance au k-ième plus proche voisin. Si on appelle X_k le k-ième plus proche voisin de x_0 :

$$\mathcal{N}_0 = \{i \in \{1, \ldots, n\} : d(X_i, x_0) \leq d(x_0, X_k)\}$$

Il estime ensuite la probabilité conditionnelle d'appartenir à la classe j comme la fraction des points de $\mathcal{N}_0$ dont la valeur de réponse est égale à j :

$$\widehat{Pr}(Y = j \mid X = x_0) = \frac{1}{k}\sum_{i \in \mathcal{N}_0} \delta_{y_i,j} \qquad \text{où } \delta_{Y,j} = \begin{cases} 1, & Y = j \\ 0, & sinon \end{cases}$$

Dans ce cours, nous utiliserons principalement la distance euclidienne (celle explicitement au programme), mais d'autres distances seront évoquées par la suite.

Enfin, la méthode KNN affecte l'observation de test x_0 à la classe dont la probabilité estimée (par l'expression précédente) est la plus élevée. En l'absence d'autres informations, on peut choisir $k = \lfloor\sqrt{n}\rfloor$, où n est la taille du jeu de données.

Notons que dans l'algorithme des k plus proches voisins, il n'y a pas de phase d'apprentissage : aucun paramètre n'est appris à partir des données. On dit que k est un hyperparamètre. Et pour éviter d'avoir deux classes de même cardinal, on choisit souvent pour k un nombre impair.

La figure ci-contre fournit un exemple illustratif de l'approche KNN. Sur cette figure est représenté un petit jeu de données d'apprentissage composé de six observations bleues et de six observations orange. L'objectif est de faire une prédiction pour le point indiqué par la croix noire. Supposons que nous choisissions k = 3. KNN identifie alors d'abord les trois observations les plus proches de la croix. Ce voisinage est représenté par un cercle (distance euclidienne). Il contient deux triangles et un cercle, ce qui conduit à des probabilités estimées de 2/3 pour la classe triangle et de 1/3 pour la classe cercle. Par conséquent, KNN prédit que la croix noire appartient à la classe triangle.

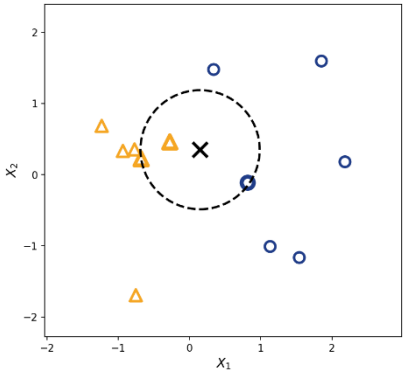

Figure 4 : Approche KNN avec k = 3

II.2. Frontière de décision

Malgré le fait qu'il s'agisse d'une approche très simple, KNN peut souvent produire des classificateurs étonnamment proches du classificateur de Bayes optimal. La Figure 5 présente la frontière de décision KNN (en trait plein noir), avec k=31, appliquée à notre jeu de données simulées. On remarque que, même si la vraie distribution n'est pas connue du classificateur KNN, la frontière de décision KNN est très proche de celle du classificateur de Bayes. Pour tracer une frontière de décision en 2D, on évalue le classifieur sur une grille de points (x_1, x_2) couvrant le domaine d'intérêt, ici [-3,3]×[-3,3].

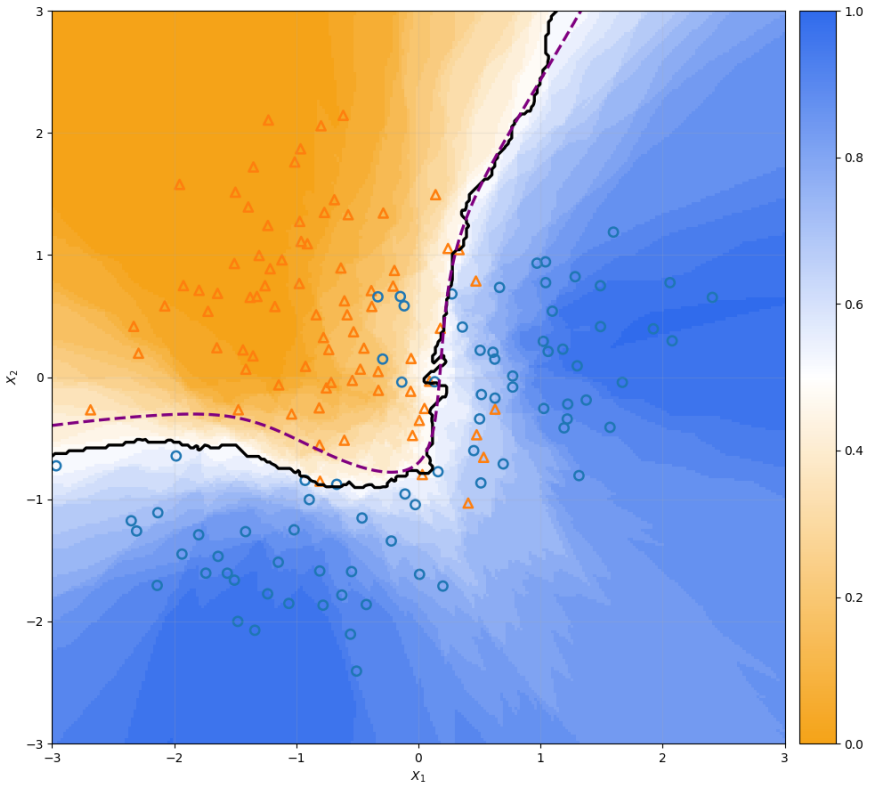

Figure 5 : Frontière de décision KNN (k=31, distance euclidienne)

Le choix de k a un effet sur le classificateur KNN obtenu. La Figure 6 présente deux ajustements KNN sur les données simulées, avec k=1 et k=100.

Lorsque k=1, la frontière de décision est excessivement flexible et détecte dans les données des structures qui ne correspondent pas à la frontière de décision de Bayes. Si on changeait légèrement les données d'entraînement, la frontière changerait beaucoup car la frontière est dentelée et dépend fortement du tirage.

À mesure que k augmente, la méthode devient moins flexible et produit une frontière de décision proche d'une frontière linéaire. Sur ce jeu de données simulées, ni k=1 ni k=100 ne donnent de bonnes prédictions.

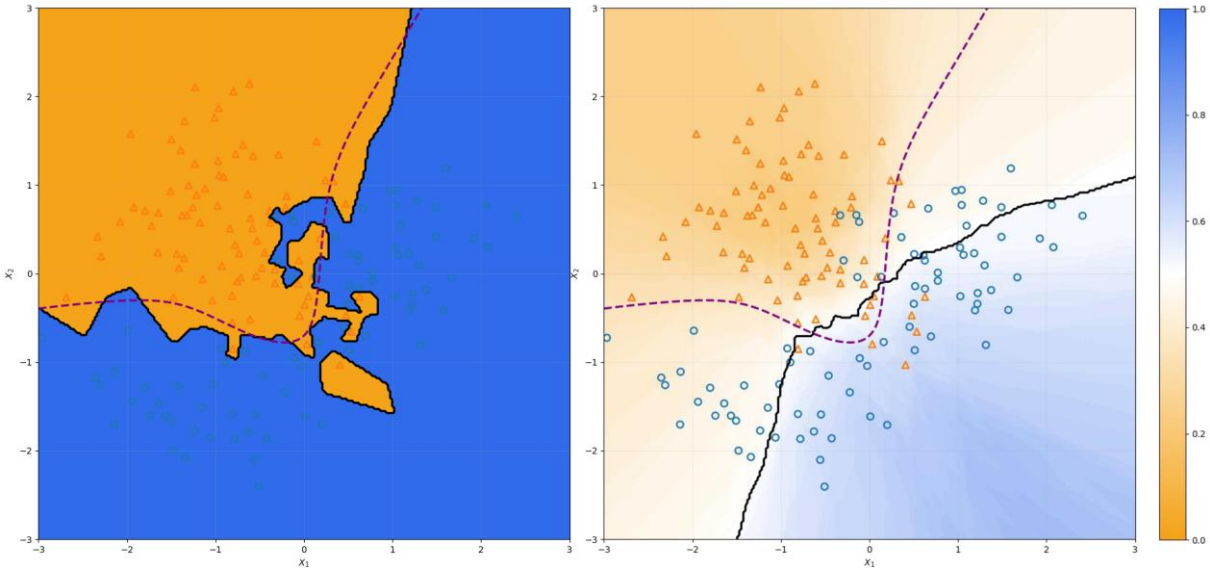

Figure 6 : Frontière de décision KNN vs Bayes (k=1 et k=100)

II.3. Erreur de généralisation : biais et variance du modèle

Nous avons vu en page 7 que l'erreur quadratique moyenne de prédiction est composée d'une erreur réductible et d'une erreur irréductible (page 7) :

$$\mathbb{E}\left[\left(Y-\hat{f}(X)\right)^2\right] = \underbrace{\mathbb{E}\left[\left(f(X)-\hat{f}(X)\right)^2\right]}_{Réductible} + \underbrace{Var(\varepsilon)}_{Irréductible}$$

Ici, f est une fonction fixe mais inconnue de X = (X_1, X_2, …, X_p) et ε est un terme d'erreur aléatoire, indépendant de X et de moyenne nulle. On avait supposé la fonction de prédiction $\hat{f}$ et l'ensemble des prédicteurs X tous deux fixés, de sorte que la seule variabilité provienne de l'erreur aléatoire ε.

On peut maintenant aller plus loin : en pratique, la fonction apprise $\hat{f}$ n'est pas fixe, car elle dépend du jeu d'entraînement : si on répète l'apprentissage sur un autre échantillon, on obtient en général un modèle différent. Nous allons introduire *l'erreur de généralisation*, qui doit donc intégrer cette nouvelle source de variabilité. Cela va conduire à décomposer l'erreur réductible en deux contributions : le biais du modèle (erreur systématique) et sa variance (sensibilité aux données).

On note $\mathbb{E}_{\mathcal{D}}$ l'espérance sur les jeux de données $\mathcal{D}$ et on suppose toujours $Y = f(X) + \varepsilon$ et $\mathbb{E}[\varepsilon] = 0$. Pour un x fixé, $\hat{f}(x)$ dépend du jeu d'entraînement $\mathcal{D}$ et on peut montrer que l'erreur quadratique moyenne sur la prédiction peut se décomposer en trois termes :

$$\mathbb{E}_{\mathcal{D}}\left[\left(Y - \hat{f}(x)\right)^2 \mid X = x\right] = \underbrace{\left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2}_{biais^2} + \underbrace{Var\left(\hat{f}(x)\right)}_{variance} + \underbrace{Var(\varepsilon)}_{irreductible}$$

Dans le contexte KNN (et plus largement en apprentissage statistique), le biais et la variance décrivent deux composantes de l'erreur de généralisation liées à l'apprentissage à partir d'un échantillon d'entraînement :

- La moyenne des modèles $\mathbb{E}[\hat{f}]$ peut être systématiquement décalée par rapport à f : c'est le biais ;
- Le modèle $\hat{f}$ change si on change l'échantillon d'apprentissage. C'est cette variabilité qui engendre la variance.

Sur la Figure 7, on fixe le modèle, puis on tire 20 jeux d'entraînement différents (80 triangles + 80 cercles). Pour chaque jeu, on calcule la frontière KNN et on superpose les 20 frontières. On compare ensuite k=1 et k=50. On ajoute la frontière de Bayes en pointillés violets.

Pour k=1, on observe davantage de frontières « dentelées » qui changent beaucoup d'un tirage à l'autre. C'est la signature d'une variance élevée : le modèle dépend fortement des points exacts de l'échantillon.

Pour k=50, les 20 frontières sont beaucoup plus regroupées : la variance est plus faible. Mais elles peuvent être systématiquement décalées par rapport à la frontière de Bayes. Ce décalage « systématique » correspond au biais : la règle est trop lissée pour suivre la vraie structure locale.

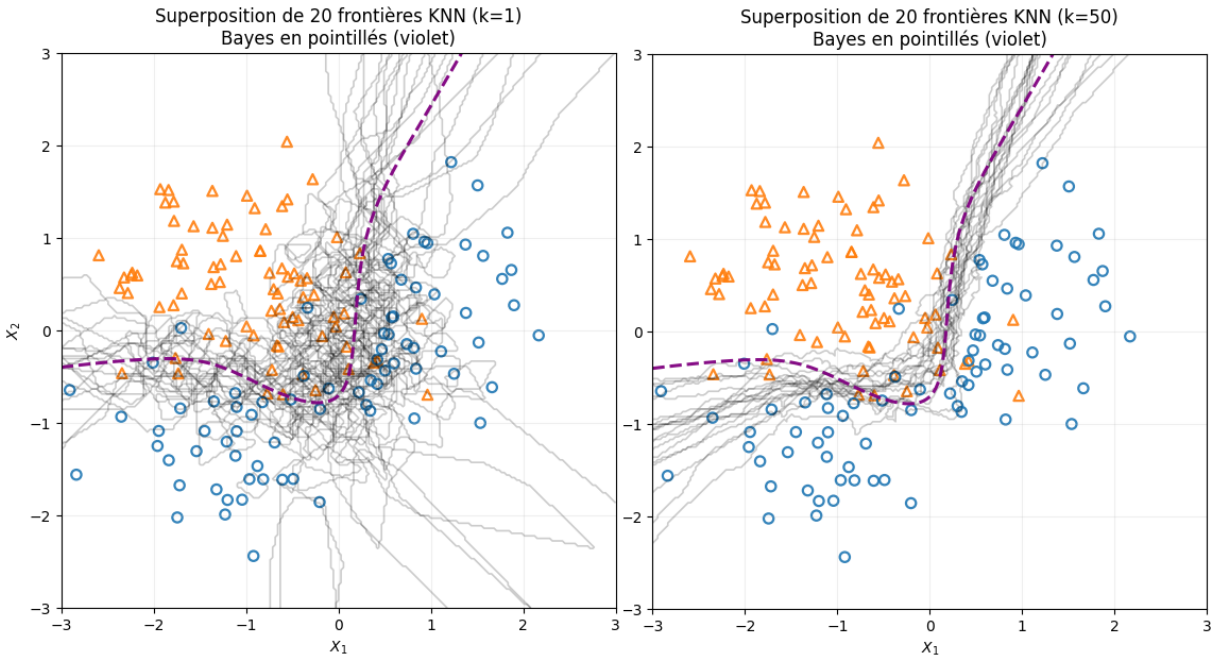

Figure 7 : Illustration de l'effet de k sur le biais et la variance du modèle

II.4. Taux d'erreur d'apprentissage et de validation

Il n'existe pas de relation forte entre le taux d'erreur d'apprentissage et le taux d'erreur de test. Avec k=1, le taux d'erreur d'apprentissage de KNN est nul, mais le taux d'erreur de test peut être assez élevé. De manière générale, lorsque l'on utilise des méthodes de classification plus flexibles, le taux d'erreur d'apprentissage diminue, mais le taux d'erreur de test ne diminue pas nécessairement.

La figure représente les erreurs de test et d'apprentissage de KNN en fonction de 1/k. Lorsque 1/k augmente, la méthode devient plus flexible. Le taux d'erreur d'apprentissage diminue systématiquement à mesure que la flexibilité augmente. En revanche, l'erreur de test présente une forme caractéristique en U : elle diminue d'abord (avec un minimum aux alentours de K=31), puis augmente de nouveau lorsque la méthode devient excessivement flexible et sur-ajuste les données.

Quand k diminue, le modèle devient plus flexible : il « colle » davantage aux données. À l'extrême, k=1 mémorise l'échantillon : chaque point est son propre plus proche voisin, donc l'erreur d'apprentissage tombe à zéro. Cela ne signifie pas que le modèle généralise bien.

L'erreur de test a souvent une forme en U car deux effets antagonistes coexistent (biais – variance). Lorsque k est grand (donc 1/k petit), la décision est très lissée et le modèle est rigide. Le biais est élevé, ce qui entraîne un sous-ajustement et une erreur de test élevée. Lorsque k est très petit (donc 1/k grand), la décision est très instable et la variance élevée. Cela entraîne un surajustement (*overfitting*) et les erreurs de test remontent.

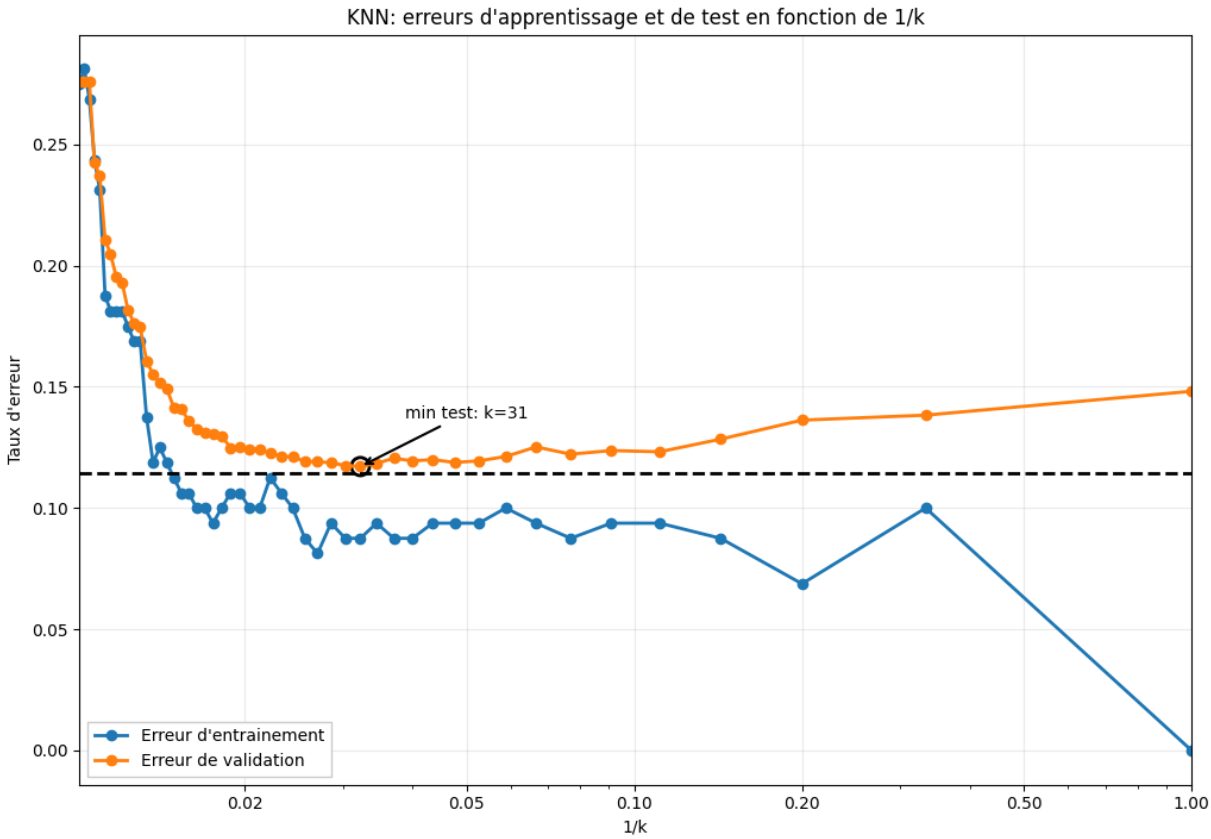

Figure 8 : Erreur d'apprentissage et de test en fonction de 1/k

II.5. Métriques géométriques usuelles

Changer la distance utilisée pour regrouper les voisins d'une observation peut changer fortement la frontière. Voici par exemple trois distances naturelles qui peuvent être utilisées avec KNN dès que les observations sont des vecteurs réels. Ces distances sont sensibles à l'échelle des variables : il faut souvent normaliser les données.

Pour deux points $x = (x_1, \ldots, x_p)$ et $z = (z_1, \ldots, z_p)$ dans $\mathbb{R}^p$:

- Distance L_1 (Manhattan) :

$$d_1(x,z) = \|x - z\|_1 = \sum_{j=1}^{p} |x_j - z_j|$$

- Distance L_2 (Euclidienne) :

$$d_2(x,z) = \|x - z\|_2 = \sqrt{\sum_{j=1}^{p} (x_j - z_j)^2}$$

- Distance L_∞ (Chebyshev) :

$$d_\infty(x,z) = \|x - z\|_\infty = \max_{1 \le j \le p} |x_j - z_j|$$

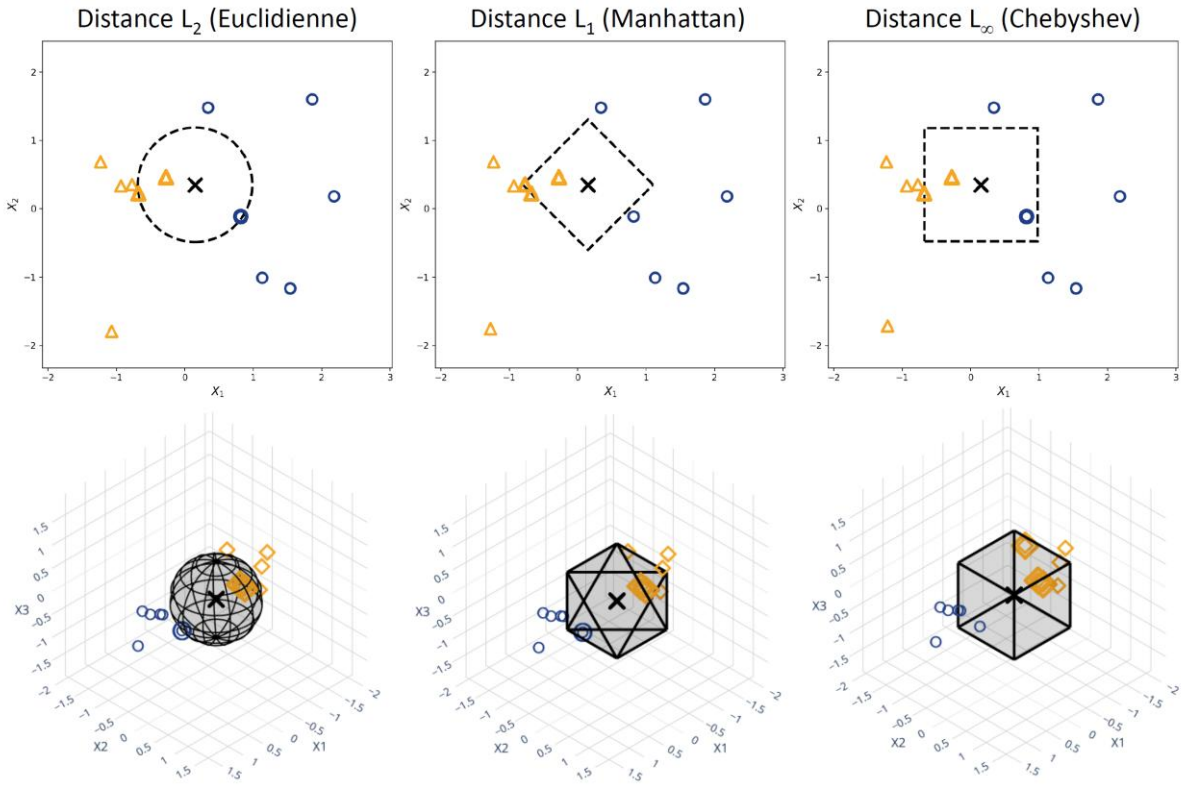

Figure 9 : Distances L_2, L_1 et L_∞

Ces distances usuelles sont utilisées dans les problèmes numériques ou qui ont une modélisation géométrique, mais elles ne sont pas adaptées à toutes les situations. On peut alors utiliser d'autres mesures ou indices de similarité dont quelques exemples sont donnés dans les paragraphes suivants.

II.6. Indice et distance de Jaccard

On définit *l'indice de Jaccard* comme une mesure de similarité entre deux ensembles finis A et B (l'un d'eux étant non vide) en posant :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \; qui\; a\; les\; propriétés \begin{cases} 0 \leq J(A,B) \leq 1 \\ J(A,B) = 1 \Leftrightarrow A = B \\ J(A,B) = 0 \Leftrightarrow A \cap B = \emptyset \end{cases}$$

On peut lui associer la *distance de Jaccard*, définie par :

$$d_J(A,B) = 1 - J(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|} = \frac{|A \,\Delta\, B|}{|A \cup B|}$$

… où $A \,\Delta\, B$ désigne la différence symétrique entre A et B : $A \,\Delta\, B = (A \cup B) \setminus (A \cap B)$

Cet indice intervient lorsqu'on souhaite comparer des éléments caractérisés par des attributs booléens/binaires (présence ou non de certaines propriétés).

Supposons que nous voulions comparer des individus selon qu'ils possèdent ou non des propriétés P_i, i = {0,1,2,3,4}. Chaque individu est représenté par un vecteur de $\{0,1\}^n$ avec $x_i = 1$ si et seulement si $P_i(x)$:

$$X = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \leftarrow \text{ne vérifie pas } P_2, \qquad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \leftarrow \text{vérifie } P_2$$

Ici, $A = \{0,1,3\}$ et $B = \{1,2\}$ donc $A \cap B = \{1\}$ et $A \cup B = \{0,1,2,3\}$ et l'indice de Jaccard vaut :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{1}{4} = 0{,}25$$

La distance de Jaccard vaut :

$$d_J(A,B) = 1 - J(A,B) = 0{,}75$$

La distance de Jaccard est nulle lorsque les vecteurs X et Y sont identiques, égale à 1 lorsque $\forall i, x_i \neq y_i$.

II.7. Comparaison de textes, matrices termes-documents

Pour évaluer la proximité entre documents textes issus d'un même corpus on commence par en donner une représentation sommaire en repérant les mots utiles du corpus et en associant à chaque document $\mathrm{D_j}$ un vecteur colonne $\mathrm{X_j}$ dont le terme $\mathrm{X_{i,j}}$ dépend de la fréquence du mot ou du terme $\mathrm{t_i}$ dans le document et dans le corpus $\mathcal{C}$.

Le nombre d'occurrences du terme pourrait servir à comparer grossièrement des documents de même taille, mais on préfère le plus souvent le codage TF-IDF (TF : fréquence ou plutôt

nombre d'occurrences des termes dans les documents, IDF : inverse du nombre de documents contenant un terme) défini par les formules :

$$x_{i,j} = \underbrace{tf_{t_i,d_j}}_{TF} \times \underbrace{ln\frac{|\mathcal{C}|}{df_{t_i}}}_{IDF} \; avec \begin{cases} tf_{t_i,d_j} : fréquence\ de\ t_i\ dans\ le\ document\ D_j \\ df_{t_i} \;\; : nombre\ de\ documents\ contenant\ t_i \end{cases}$$

Un indice de similarité entre deux documents est alors donné par la similarité cosinus définie comme le cosinus de l'angle entre $\mathrm{X_j}$ et $\mathrm{X_k}$ comme vecteurs de $\mathbb{R}^n$ avec n la taille du lexique retenu (de quelques milliers à quelques dizaines de milliers de termes en pratique) :

$$sim(D_j, D_k) = \frac{\sum_i x_{i,j}x_{i,k}}{\|X_j\|_2\|X_k\|_2} = \frac{\langle X_j, X_k\rangle}{\|X_j\|_2\|X_k\|_2}$$

Dans le cas où le terme $\mathrm{t_i}$ est présent dans tous les documents, on a alors $df_{t_i} = |\mathcal{C}|$, et donc son IDF vaut 0 et $\mathrm{x_{i,j}} = 0$. Ce serait le cas avec les mots courants de la langue, articles, prépositions, auxiliaires que l'on évite systématiquement de prendre en compte (mots-vides ou *stop-words* en anglais).

Dans le cas où le terme $\mathrm{t_i}$ n'est présent que dans un seul document $\mathrm{D_j}$, on a $df_{t_i} = 1$. Ainsi le coefficient de tf_{t_i,d_j} dans $x_{i,j}$ prend la valeur maximale et $x_{i,j} = tf_{t_i,d_j} ln|\mathcal{C}|$.

Si on pose $\delta(\mathrm{D_j}, \mathrm{D_k}) = 1 - sim(D_j, D_k)$, alors si $\delta(\mathrm{D_j}, \mathrm{D_k}) = 0$ cela signifie que $sim(D_j, D_k) = 1$ et donc que les deux vecteurs sont proportionnels. Donc pour tout indice i qui correspond au même terme $\mathrm{t_i}$, on a :

$$k = \frac{x_{i,j}}{x_{i,k}} = \frac{tf_{t_i,d_j} \times ln\frac{|\mathcal{C}|}{df_{t_i}}}{tf_{t_i,d_k} \times ln\frac{|\mathcal{C}|}{df_{t_i}}} = \frac{tf_{t_i,d_j}}{tf_{t_i,d_k}}$$

Cela signifie que les occurrences des mots dans les deux documents sont proportionnelles, comme par exemple si D_j = « Bonjour à toi. » et D_k = « Bonjour à toi. À toi, Bonjour ! ».

Prenons l'exemple suivant composé de trois documents : D_1 = « chat mange poisson » ; D_2 = « chien mange os » et D_3 = « chat chat mange ».
Le vocabulaire associé est : [chat, chien, mange, poisson, os].

Les fréquences des termes : $tf_{t_i,d_1} = (1,0,1,1,0)$, $tf_{t_i,d_2} = (0,1,1,0,1)$ et $tf_{t_i,d_3} = (2,0,1,0,0)$
Nombre de documents contenant les termes : $df_{t_i} = (2,1,3,1,1)$

$$x_1 = tf_{t_i,d_1} \times ln\frac{|\mathcal{C}|}{df_{t_i}} = tf_{t_i,d_1} \times ln\frac{3}{(2,1,3,1,1)}$$
$$= tf_{t_i,d_1} \times (0.405, 1.0986, 0, 1.0986\ , 1.0986)$$
$$= (1,0,1,1,0) \cdot (0.405, 1.0986, 0, 1.0986\ , 1.0986) = (0.405, 0, 0, 1.0986, 0)$$
$$x_2 = (0,1,1,0,1) \cdot (0.405, 1.0986, 0, 1.0986\ , 1.0986) = (0, 1{,}0986, 0, 0, 1.0986)$$
$$x_3 = (2,0,1,0,0) \cdot (0.405, 1.0986, 0, 1.0986\ , 1.0986) = (0.8109, 0, 0, 0, 0)$$

- $sim(D_1, D_2) = \frac{\langle X_1, X_2\rangle}{\|X_1\|_2\|X_2\|_2} = 0$: le seul mot commun est « mange » mais comme il est présent partout, son IDF vaut 0 et il ne contribue pas au TF-IDF.
- $sim(D_1, D_3) = 0.345$; $sim(D_2, D_3) = 0$: le seul mot commun est « mange »

II.8. Algorithme KNN en classification

On utilisera ici les notations argmax et argmin : étant donné des réels $a_1, \ldots, a_p$, la notation $\underset{j \in [\![1,p]\!]}{\operatorname{argmax}} a_j$ ou $\operatorname{argmax}(a_1, \ldots, a_p)$ désigne un indice $j_0 \in [\![1,p]\!]$ tel que $\max_{j \in [\![1,p]\!]} a_j = a_{j_0}$. La notation argmin est définie de manière analogue.

On considère une classification multi-classes avec K classes c_1, c_2, …, c_p, un ensemble de test $\{(x_i, y_i)\}_{i=1..n}$.

Algorithme des k plus proches voisins - classification

KNN_CLASSIFICATION(x) :

Renvoyer $\underset{j \in [\![1,p]\!]}{\operatorname{argmax}} |\{i \in [\![1,n]\!] : x_i \in \mathcal{N}_k(x) \text{ et } y_i = j\}|$

Voici une implémentation en Python avec distance euclidienne :

```python
import numpy as np
    """
    X : array-like, shape (n, p)
        Données d'entraînement (n points en dimension p).
    y : array-like, shape (n,)
        Étiquettes d'entraînement.
    z : array-like, shape (p,)
        Point à classer.
    k : int
        Nombre de plus proches voisins (1 <= k <= n).

    y_pred : même type que y, classe prédite pour z.
    """
def knn_classification(X, y, z, k):
    X = np.asarray(X)
    y = np.asarray(y)
    z = np.asarray(z)

    # Normes euclidiennes
    d2 = np.linalg.norm((X - z), axis=1)     # shape (n,)

    # Indices des k plus proches voisins
    idx = np.argsort(d2)[:k]                 # shape (k,)

    # Labels des voisins
    voisins = y[idx]

    # Vote majoritaire
    classes, counts = np.unique(voisins, return_counts=True)

    # En cas d'égalité, np.argmax choisit la première occurrence
    return classes[np.argmax(counts)]
```

Parmi les fonctions de la bibliothèque Numpy utilisées :

- `np.asarray(...)` : permet de convertir une liste Python en tableau NumPy
  `X = np.asarray(X)` transforme par exemple une liste de n points [[...],[...],...] en tableau 2D de forme (n, p).
- `np.argsort(…)` : ne trie pas les valeurs ; retourne les indices qui trieraient le tableau.
  Exemple : si d2 = [0.3, 2.1, 0.1], alors np.argsort(d2) = [2, 0, 1]
- `np.unique(voisins, return_counts=True)` : renvoie les valeurs distinctes triées et éventuellement combien de fois cette valeur apparait.
  Exemple : voisins = [1,1,0,1,0], classes = [0,1] et counts = [2, 3]
- `np.argmax(counts)` : renvoie l'indice de la plus grande valeur.
  Exemple : counts=[2,3] → np.argmax(counts)=1

Concernant `np.linalg.norm(..., axis=1)` : Calcul la norme sur un axe spécifique.
Si x est (n,p) et z est (p,) alors axis=1 donne un résultat (n,) (norme ligne par ligne)
Exemple : X = np.array([[1,2,3],[4,5,6]]) possède une forme (2,3)
np.linalg.norm(X, axis=0) renvoie une forme (3,) : [4.1, 5.4, 6.7]
np.linalg.norm(X, axis=1) renvoie une forme (2,) : [3.7, 8.8]

Dans le programme, np.linalg.norm((X – z), axis=1)**2 réalise l'opération de cette manière :

- X a une forme (n, p) : n points de dimension p, forme (n,p)
  - Exemple : X = np.array([[1,2],[3,4],[5,6]]) : Trois points de dimension 2 (3,2)
- z est le point à classer, de dimension p, forme (p,)
  - Exemple : z = np.array([3,4]) : Point de dimension 2 (2,)

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad ; \; z = [3 \quad 4]$$

- (X - z) : soustrait les composantes de z à chaque point contenu dans x ; c'est un array de forme (3,2)

$$X - z = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} - [3 \quad 4] = \begin{bmatrix} x_1 - z \\ x_2 - z \\ x_3 - z \end{bmatrix} = \begin{bmatrix} -2 & -2 \\ 0 & 0 \\ 2 & 2 \end{bmatrix}$$

- np.linalg.norm((X - z), axis=1) : La fonction calcule la norme sur l'axe 1 (norme ligne par ligne). Il retourne un résultat de forme (3,) : [2.83, 0, 2.83]

$$np.linalg.norm\left(\begin{bmatrix} -2 & -2 \\ 0 & 0 \\ 2 & 2 \end{bmatrix}, axis = 1\right) = \begin{bmatrix} \|(-2,-2)\|_2 \\ \|(0,0)\|_2 \\ \|(2,2)\|_2 \end{bmatrix} = \begin{bmatrix} \|x_1 - z\|_2 \\ \|x_2 - z\|_2 \\ \|x_3 - z\|_2 \end{bmatrix}$$

On peut voir quelques fois que la norme est élevée au carré (pour respecter l'écriture usuelle en math $\|\cdot\|^2$). Cela peut se faire avec l'opération **2 qui élève chaque élément au carré (élément par élément) :

$$np.linalg.norm\left(\begin{bmatrix} -2 & -2 \\ 0 & 0 \\ 2 & 2 \end{bmatrix}, axis = 1\right) ** 2 == \begin{bmatrix} \|x_1 - z\|_2{}^2 \\ \|x_2 - z\|_2{}^2 \\ \|x_3 - z\|_2{}^2 \end{bmatrix} = \begin{bmatrix} 8 \\ 0 \\ 8 \end{bmatrix}$$

II.9. Algorithme KNN en régression

On considère une régression avec un ensemble d'apprentissage $\{(x_i, y_i)\}_{i=1..n}$.

Pour un problème de régression, on associe à x la moyenne des étiquettes de ses k plus proches voisins.

Algorithme des k plus proches voisins - régression

KNN_REGRESSION(x) :

Renvoyer $\frac{1}{k} \sum_{\substack{i=1 \\ x_i \in \mathcal{N}_k(x)}}^{n} y_i$

Voici une implémentation en Python avec distance euclidienne :

```python
import numpy as np
    """
    X : array-like, shape (n, p)
        Données d'entraînement (n points en dimension p).
    y : array-like, shape (n,)
        Étiquettes d'entraînement.
    z : array-like, shape (p,)
        Point à classer.
    k : int
        Nombre de plus proches voisins (1 <= k <= n).

    y_pred : même type que y, valeur prédite pour z.
    """
def KNN_REGRESSION(x,y,z,k):
    x=np.array(x)
    y=np.array(y)

    # Liste des normes au carré des z-x[i]
    N=np.linalg.norm((z-x),axis=1)**2

    # Indices des k plus proches voisins
    indices_KNN=np.argsort(N)[:k]

    voisins=np.array([y[i] for i in indices_KNN])

    return np.mean(voisins)
```