

VI Algorithme des K plus proches voisins

3 juillet 2024

1 Introduction

1.1 Pré-traitement

Les méthodes d'apprentissage automatique sont construites pour fonctionner sur des données vectorielles (des vecteurs de \mathbb{R}^d , pour un d particulier), où la dimension des objets (ici d) est « raisonnable » au vu du nombre de données n (voir la « malédiction de la dimension », *curse of dimensionality* en anglais). Or, les données que l'on veut traiter sont le plus souvent :

- soit non vectorielles (exemple : vidéos, textes) ;
- soit vectorielles, mais de dimensions trop élevées (exemple : grandes images).

La première étape d'un système d'apprentissage automatique consiste donc souvent à transformer les données « brutes » en données vectorielles, via ce que l'on appelle une *fonction de description*. Plus précisément, si l'on note \mathcal{O} l'ensemble des objets sur lesquels on va travailler, on construit une fonction $f : \mathcal{O} \rightarrow \mathbb{R}^d$ vérifiant deux conditions :

- d n'est pas trop grand par rapport au nombre d'observations n ;
- si deux objets o, o' sont semblables, au sens du problème étudié, alors $f(o)$ et $f(o')$ devront être « proches », en un certain sens.

Construire une telle fonction f est une étape fondamentale d'un système de classification, souvent ardue, et spécifique à chaque problème. Nous ne nous y intéresserons pas, et nous supposerons que l'on travaille directement dans \mathbb{R}^d (nous prendrons $d = 2$ pour les illustrations). On travaillera aussi avec la distance euclidienne sur \mathbb{R}^d .

1.2 Classification

On s'intéresse dans ce chapitre à un problème classique d'apprentissage automatique : la *classification*.

On considère que l'on observe n objets X_0, \dots, X_{n-1} appartenant à un ensemble \mathbb{R}^d . À chaque observation X_i , on a attribué une étiquette e_i , les étiquettes appartenant

à un ensemble d'étiquettes \mathcal{E} . On appelle cela *l'échantillon*, que l'on pourra noter

$$\mathcal{S} = \{(X_0, e_0), \dots, (X_{n-1}, e_{n-1})\}.$$

On nous donne un nouvel objet X , non étiqueté. La tâche de classification est la suivante : déterminer comment étiqueter « au mieux » ce nouvel objet (on dit aussi : *classifier* ce objet), à la vue de l'échantillon.

Bien entendu, cette tâche est bien trop abstraite et générale pour pouvoir être traitée telle quelle en toute généralité. Nous allons voir une méthode particulière permettant de proposer une solution à ce problème dans un cas particulier.

Exemple 1.2.1 (détection de personnes).

Les objets peuvent être des photographies, avec deux étiquettes : « il y a un visage humain sur la photographie », « il n'y a pas de visage humain sur la photographie ». Les étiquettes sont ici binaires.

Exemple 1.2.2 (détection de visages).

Les objets peuvent être des photographies personnes connues (une étiquette par personne). Il y a autant d'étiquettes que de personnes différentes dans l'échantillon.

Exemple 1.2.3 (détection de caractères).

Les objets peuvent être de petites images censées provenir d'un texte (par exemple scanné), avec une étiquette par caractère de l'alphabet utilisé. Il y a autant d'étiquettes que de caractères.

Exemple 1.2.4 (détection de spams).

Les objets peuvent être de emails, avec deux étiquettes : « ce message est un spam », « ce message n'est pas un spam ». Là encore, les étiquettes sont binaires.

Remarque 1.2.5.

Créer cet échantillon, et surtout étiqueter les données, nécessite un travail particulier, souvent réalisé à la main. Nous ne nous intéresserons pas à cette question, ni à celle de la qualité de ces données : nous supposerons que l'échantillon donné est parfait.

On parle notamment ici *d'apprentissage supervisé*, la supervision mentionnée provenant notamment de la création de l'échantillon, et surtout son étiquetage.

1.3 Rappels de géométrie élémentaire de \mathbb{R}^d

On rappelle que pour deux vecteurs

$$X = \begin{pmatrix} x_0 \\ \vdots \\ x_{d-1} \end{pmatrix}, Y = \begin{pmatrix} y_0 \\ \vdots \\ y_{d-1} \end{pmatrix} \in \mathbb{R}^d,$$

la distance euclidienne entre ces deux vecteurs X, Y est

$$d(X, Y) = \sqrt{\sum_{i=0}^{d-1} (x_i - y_i)^2} = \sqrt{(X - Y)^\top (X - Y)}.$$

En supposant que les opérations sur les flottants (somme, carré, racine carrée) se font en temps $\mathcal{O}(1)$, il est clair que l'on peut calculer la distance $d(X, Y)$ en temps $\mathcal{O}(d)$.

2 L'algorithme des K plus proches voisins

2.1 Description et pseudo-code

C'est un algorithme des plus simples : on considère des points $X_0, \dots, X_{n-1} \in \mathbb{R}^d$ préalablement étiquetés, et un nouveau point $X \in \mathbb{R}^d$.

On détermine alors les K éléments parmi $[X_0, \dots, X_{n-1}]$ les plus proches de X , et l'on affecte à X l'étiquette majoritaire parmi ces points.

Remarque 2.1.1.

Bien entendu, on demande à ce que $K \leq n$, et il faut déterminer ce qui se passe s'il y a égalité entre deux étiquettes « à égalité ».

Nous considérerons que l'échantillon est donné sous la forme d'une liste de couples (point, étiquette).

Il faut d'abord savoir calculer la distance entre deux points, ce qui se fait assez simplement avec une boucle (on peut en donner une écriture très concise et sans boucle en Python, à vous de le faire).

```
Fonction d(p1, p2) :
    d ← dimension de p1
    s ← 0
    Pour i variant de 0 à d-1, Faire
        | s ← s + (p1[i] - p2[i])2
    Renvoyer √s
Fin Fonction
```

Remarque 2.1.2.

Dans ce chapitre, on utilise cette distance uniquement à but de comparaison. On comparera donc plutôt les carrés des distances, ce qui évite de calculer une racine carrée.

Remarque 2.1.3.

Il y a de nombreuses manières de présenter l'échantillon. On peut par exemple supposer que l'on a une liste de couples de la forme (vecteur, étiquette), ou bien deux listes : une de vecteurs, et une d'étiquettes.

Nous nous situerons dans le deuxième cas. On considérera que sont données deux listes :

- `listeX` : la liste des vecteurs $[X_0, \dots, X_{n-1}]$;
- `listeEtiquettes` : la liste des étiquettes $[e_0, \dots, e_{n-1}]$, où e_i est l'étiquette la donnée X_i .

Il faut ensuite savoir trier l'échantillon en fonction de la distance à un point donné. Voici une solution utilisant le tri rapide. Est donné en plus un vecteur Y : on trie l'échantillon par distance croissante à Y .

Fonction TRI_RAPIDE(listeX, listeEtiquettes, Y) :

Si listeX a au plus 1 élément **Alors**

 |Renvoyer listeX, listeEtiquettes

Sinon

 piv \leftarrow listeX[0]

 1X1 \leftarrow [x \in listeX[1 :] tq. $d(x, Y) \leq d(piv, Y)$]

 1X2 \leftarrow [x \in listeX[1 :] tq. $d(x, Y) > d(piv, Y)$]

 1E1 \leftarrow [x \in listeEtiquettes[1 :] tq. $d(x, Y) \leq d(piv, Y)$]

 1E2 \leftarrow [x \in listeEtiquettes[1 :] tq. $d(x, Y) > d(piv, Y)$]

 1X1, 1E1 \leftarrow TRI_RAPIDE(1X1, 1E1)

 1X2, 1E2 \leftarrow TRI_RAPIDE(1X2, 1E2)

 listeXtriee \leftarrow Concaténation de 1X1, [piv], 1X2

 listeEtriee \leftarrow Concaténation de 1E1, [listeE[0]], 1E2

 |Renvoyer listeXtriee, listeEtriee

Fin Fonction

Fonction clef_maxi(d) :

 ## Précondition : les valeurs du dictionnaire d sont des entiers positifs

 cmax $\leftarrow \emptyset$

 vmax $\leftarrow -1$

Pour chaque clef De d, **Faire**

Si d[clef] > vmax **Alors**

 | cmax \leftarrow clef

 | vmax \leftarrow d[clef]

 |Renvoyer cmax

Fin Fonction

On peut alors écrire un algorithme de vote parmi les K premiers éléments de l'échantillon (qui est censé être déjà trié).

Fonction vote(listeEtiquettesTriees, K) :

 ## Précondition : échantillon trié par distance croissante par rapport au point

 d \leftarrow dictionnaire vide

Pour chaque étiquette De K premiers éléments de listeEtiquettesTriees, **Faire**

Si étiquette est déjà une clef de d **Alors**

 | d[étiquette] incrémenté de 1

Sinon

 | d[étiquette] $\leftarrow 1$

 |Renvoyer clef_maxi(d)

Fin Fonction

On a donc tout ce qu'il faut pour écrire un algorithme des K plus proches voisins.

Fonction knn(listeX, listeEtiquettes, Y, K) :

 | _, listeEtiquettesTriees \leftarrow trie(listeX, listeEtiquettes, Y)

 | Renvoyer vote(listeEtiquettesTriees, K)

Fin Fonction

Remarque 2.1.5.

Ici, on a réalisé le vote en attribuant le même poids aux K plus proches voisins du

Exercice 2.1.4.

Écrire ce tri en utilisant l'algorithme de tri fusion.

Il suffit maintenant de réaliser un vote parmi les K premiers éléments triés par distance croissante par rapport au nouveau point que l'on veut classifier. Pour cela, on peut construire un dictionnaire des étiquettes que l'on rencontre parmi ces K premières éléments de l'échantillon : les clefs du dictionnaire seront des étiquettes, les valeurs le nombre de fois où l'on rencontre chaque étiquette. Pour cela :

- lorsque l'on rencontre une nouvelle étiquette, on initialise un compteur à 1 ;
- si l'on rencontre une étiquette déjà rencontrée, on augmente son compteur de 1.

Il suffit ensuite de parcourir ce dictionnaire et d'en sélectionner le maximum.

Pour cela, on peut déjà écrire un fonction qui prend en argument un dictionnaire dont les valeurs sont positives, et qui renvoie une clef associée à une valeur maximale.

point considéré. Il y a d'autres manières de procéder, qui consistent à attribuer un poids plus important aux points les plus proches du point considéré. Nous ne les traiterons pas ici.

Remarque 2.1.6.

Si \mathcal{E} est l'ensemble des étiquettes, on vient de construire une fonction $\hat{f} : \mathbb{R}^d \rightarrow \mathcal{E}$, appelée *classificateur* (ou fonction de classification). Cette fonction dépend bien entendu de l'échantillon sur lequel on travaille.

Exercice 2.1.7.

On considère un robot miniature se déplaçant de manière autonome en suivant un fléchage collé au sol. Ce robot est doté d'une caméra de faible résolution (16 par 16 pixels), qui lui permet de lire les étiquettes portant les flèches. L'objectif est de détecter le type de flèche lue (parmi quatre possibilités : haut, bas, gauche et droite).

Vous trouverez sur mon site un script `fleches_donnees.py` contenant un échantillon correspondant (les listes `Xtrain` et `Etrain`), ainsi que des données supplémentaires permettant d'évaluer la qualité du classificateur que vous allez construire (les listes `Xtest` et `Etest`).

- Chaque image est représentée sous forme d'un vecteur de taille 256 (en mettant bout à bout les lignes de la matrice représentant l'image).
- Chaque étiquette est une chaîne de caractères parmi `haut`, `bas`, `gauche`, `droite`.

Enfin, la fonction `affiche` prend en argument un vecteur représentant une telle image, et l'affiche.

Implémenter en Python les algorithmes vus ci dessus. Pour $K = 1$, quelle est l'étiquette prédite pour l'image `Xtest[0]` ? Et pour l'image `Xtest[1]` ? Sont-elles correctes ? Obtenir la liste des erreurs.

Vous pourrez travailler dans le script que vous avez téléchargé, ou bien commencer à travailler dans un nouveau script en commençant par la ligne suivante.

```
from fleches_donnees import *
```

2.2 Influence du paramètre K

Dans les exemples suivants, nous avons tiré des points aléatoirement dans le carré $[0, 1]^2$, et nous avons affecté à chacun une étiquette (0 pour les croix rouges \times , 1 pour les croix bleues $+$) (voir figure 1). Bien entendu, cet étiquetage ne se fait pas de manière uniforme partout sur le carré.

On calcule alors l'estimateur des K plus proches voisins pour plusieurs valeurs de K en chaque point de $[0, 1]^2$. Les points étiquetés en 0 sont en rouge, ceux étiquetés en 1 sont en bleu.

Les résultats sont donnés dans la figure 2.

La question suivante se pose naturellement : comment choisir la meilleure valeur de K ? Nous donnerons des éléments de réponse dans la partie suivante.

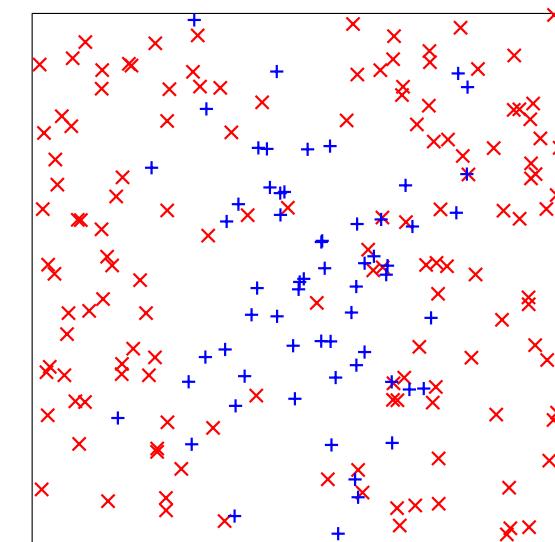
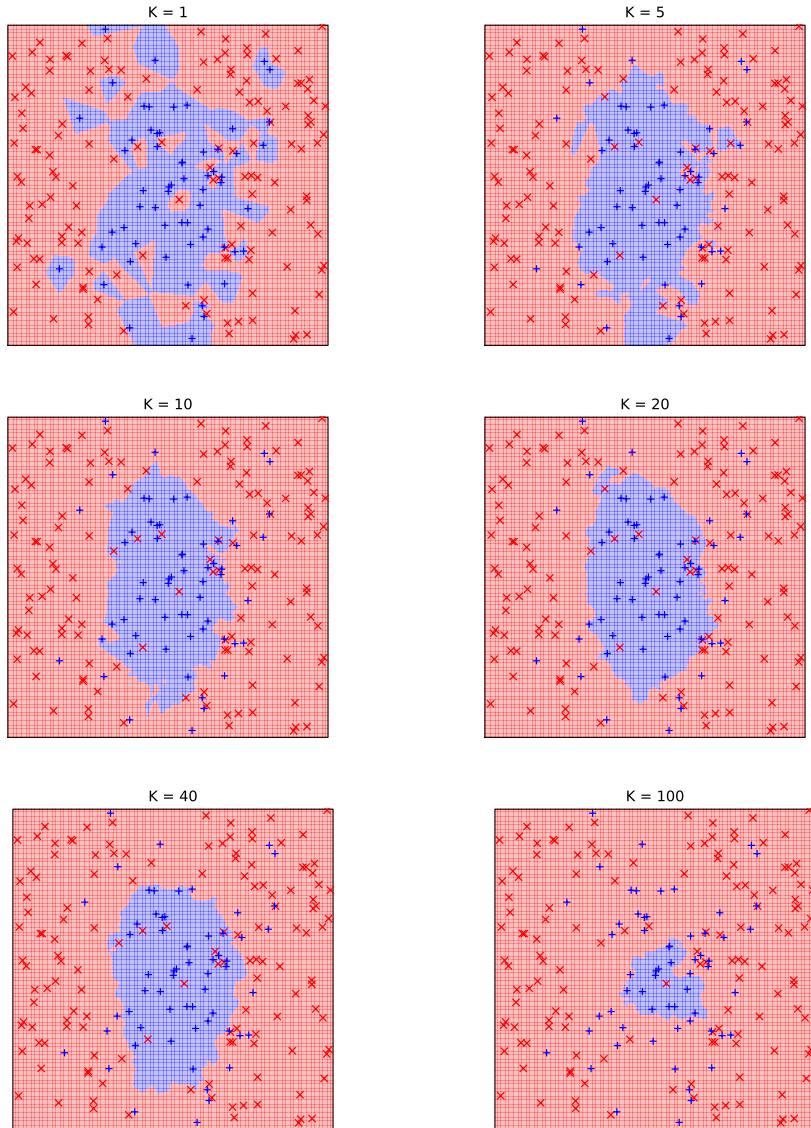


FIGURE 1 – Notre échantillon

FIGURE 2 – Classification par K plus proches voisins pour $K \in [1, 5, 10, 20, 40, 100]$

3 Évaluation de la qualité du classificateur

3.1 Erreur de classification

Une question cruciale est celle de l'évaluation de la qualité du classificateur ainsi construit. Si l'on suppose que les données (*i.e.* les couples (observation, étiquette)) sont des réalisations i.i.d. d'une loi \mathcal{P} , en notant (X, E) un tel couple et f le classificateur, un objectif peut être d'estimer (et ensuite de minimiser) *l'erreur de classification* :

$$\text{Err}(f) = P(f(X) \neq E).$$

Remarque 3.1.1.

Il existe une fonction f^* minimisant l'erreur de classification, appelée *classificateur de Bayes*.

Définition 3.1.2 (erreur de classification empirique).

L'erreur de classification empirique d'un classificateur f sur un échantillon \mathcal{S} est la proportion d'erreurs que réalise ce classificateur sur cet échantillon, *i.e.* la proportion d'observations $(X, e) \in \mathcal{S}$ vérifiant $f(X) \neq e$:

$$\widehat{\text{Err}}(f, \mathcal{S}) = \frac{1}{n} \sum_{i=0}^{n-1} \mathbf{1}_{f(X_i) \neq e_i} = \frac{\text{Card} \{ (X, e) \in \mathcal{S} \mid f(X) \neq e \}}{\text{Card}(\mathcal{S})}$$

Remarque 3.1.3.

La loi des grands nombres indique que si f est indépendant de \mathcal{S} , alors

$$\widehat{\text{Err}}(f, \mathcal{S}) \xrightarrow[n \rightarrow +\infty]{\mathbb{P}} \text{Err}(f).$$

On pourrait-être tenté de prendre comme critère de qualité $\widehat{\text{Err}}(\hat{f}, \mathcal{S})$, ce qui revient à

- considérer les observations de l'échantillon, d'appliquer dessus le classificateur que l'on a construit ;
- compter la proportion de fois où le classificateur commet une erreur, *i.e.* où le classificateur prédit une étiquette différente de celle de l'observation.

C'est toujours une TRÈS MAUVAISE IDÉE.

En effet, \hat{f} n'est pas indépendant de \mathcal{S} , car \hat{f} est construit sur \mathcal{S} ! De manière générale, $\widehat{\text{Err}}(\hat{f}, \mathcal{S})$ surestime $\text{Err}(\hat{f})$.

On retiendra par dessus tout la règle suivante.

Règle d'or

On n'évalue jamais la qualité d'un processus sur les données qui ont servi à le construire.

Il existe une manière de procéder : on divise l'échantillon en deux échantillons.

- Un premier échantillon \mathcal{S}_E servira à construire le classificateur (on applique dessus l'algorithme des K plus proches voisins) : c'est l'*échantillon d'entraînement*.
- Un deuxième échantillon \mathcal{S}_T servira à évaluer la performance du classificateur ainsi construit : c'est l'*échantillon de test*.

Traditionnellement, on réserve environ 80% des données à l'échantillon d'entraînement, et 20% des données à celui de test.

Remarque 3.1.4.

Pour sélectionner ces deux échantillon de test et d'entraînement, on prendra soin de « mélanger » préalablement les données. Cela peut se faire aisément en tirant un nombre flottant aléatoire dans $[0, 1]$ pour chaque observation, et en triant l'échantillon en fonction de ce nombre aléatoire. On permute ainsi l'échantillon de manière aléatoire. Il suffit alors de prendre les 80 premiers pourcents de l'échantillon ainsi permétré pour l'entraînement, et le reste pour le test.

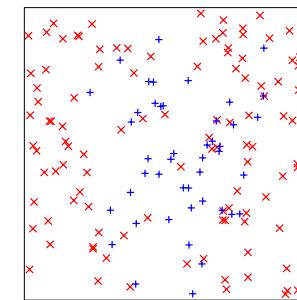
Le processus sera alors le suivant :

1. Séparation de l'échantillon en un échantillon d'entraînement et un échantillon de test (voir figure 3).
2. On entraîne le classificateur sur l'échantillon d'entraînement uniquement (voir figure 4, on montre cela pour $K = 5$ et $K = 60$) : on construit donc un classificateur \hat{f} .
3. On évalue le classificateur ainsi entraîné sur l'échantillon de test, en calculant dessus l'erreur empirique de classification $\widehat{\text{Err}}(\hat{f}, \mathcal{S}_T)$, ce qui revient à calculer la proportion des observations de \mathcal{S}_T incorrectement classifiées (voir figure 5, on montre cela pour $K = 5$ et $K = 60$). Il suffit donc de compter le nombre d'observation $(X, e) \in \mathcal{S}_T$ vérifiant $\hat{f}(X) \neq e$, et de diviser cela par le nombre d'éléments de \mathcal{S}_T .

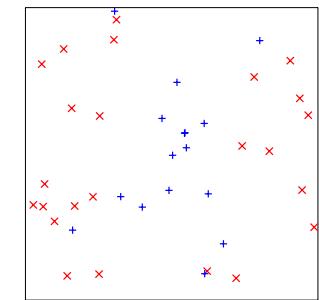
On présente dans la figure 6 le pourcentage d'erreurs (empiriques) de classification en faisant varier K . Ici, les valeurs optimales de K sont pour $K = 5$ et $K = 15$. On peut alors choisir le paramètre K minimisant cette erreur (empirique) de classification : on parle alors de *calibration* du classificateur.

Exercice 3.1.5.

On reprend l'exercice 2.1.7.



(a) Échantillon d'entraînement



(b) Échantillon de test

FIGURE 3 – Séparation de l'échantillon en un échantillon d'entraînement et un échantillon de test

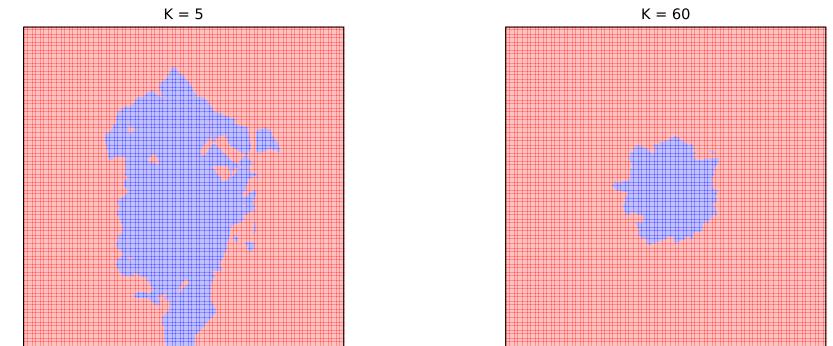
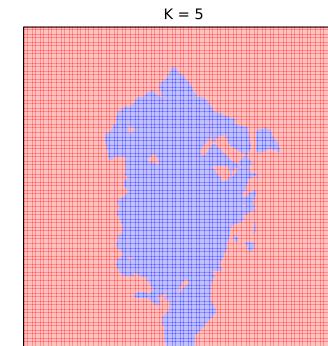


FIGURE 4 – Entraînement du classificateur sur l'échantillon d'entraînement

Calculer les erreurs de classification sur l'échantillon de test pour K allant de 1 à 20. Qu'en dites-vous ?

3.2 Matrice de confusion

De manière un peu plus fine, pour deux étiquettes E_i, E_j , on peut s'intéresser à la probabilité de classifier X en E_j sachant que X est réellement étiquetée en E_i .

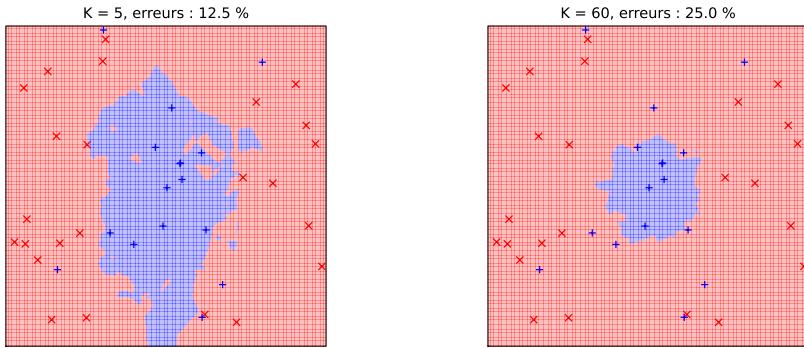


FIGURE 5 – Évaluation du classificateur entraîné sur l'échantillon de test

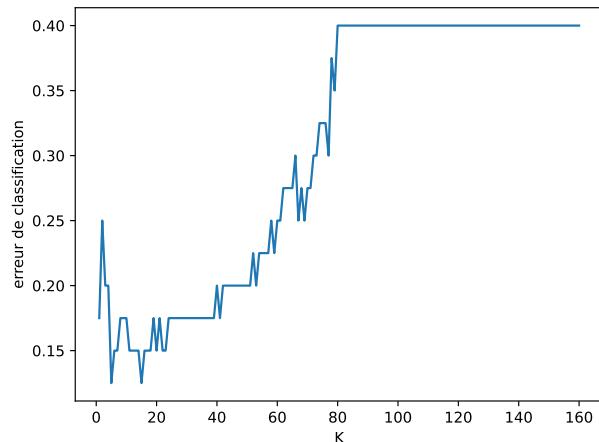


FIGURE 6 – Évolution de l'erreur de classification

Définition 3.2.1 (matrice de confusion).

On considère une tâche de classification à p étiquettes (E_0, \dots, E_{p-1}) , un classificateur f et un échantillon de test $(X_0, e_0), \dots, (X_{n-1}, e_{n-1})$. La matrice de confusion est la matrice comptant pour chaque ligne i et chaque colonne j le nombre d'observations d'étiquette E_i classifiées avec l'étiquette E_j . Plus formellement, c'est la matrice $M =$

$(M_{i,j})$ de dimension $p \times p$ telle que pour chaque $0 \leq i, j < p$:

$$M_{i,j} = \text{Card} \{ k \in \llbracket 0, n \rrbracket \mid e_k = E_i \text{ et } f(X_k) = E_j \}.$$

Remarque 3.2.2.

Pour chaque $0 \leq i, j < p$, $M_{i,j}$ compte donc le nombre d'observations d'étiquettes E_i (donc du type (X, E_i)) classifiées en E_j , *i.e.* vérifiant $f(X) = E_j$.

Remarque 3.2.3.

Étant donné la matrice de confusion M , on remarquera que :

- la taille de l'échantillon de test est la somme des éléments de M ;
- le nombre d'erreurs de classification est la somme des éléments non diagonaux de M .

On peut donc aisément calculer l'erreur de classification à partir de cette matrice.

Remarque 3.2.4.

On peut très bien inverser les rôles des lignes et des colonnes dans la matrice de confusion, il n'y a pas de convention universelle dessus. Toutefois, cela ne change pas la manière de calculer l'erreur de classification.

Exemple 3.2.5.

Pour $K = 5$, la matrice de confusion est ici

$$\begin{pmatrix} 24 & 0 \\ 5 & 11 \end{pmatrix}.$$

On lit qu'il y a 16 observations de test d'étiquettes 1 : 5 ont été classifiées en 0 (ce sont des erreurs), et 11 en 1 (ce sont les classifications correctes). Il y a 24 observations d'étiquette 0, toutes ont été correctement classifiées.

Pour $K = 60$, la matrice de confusion est

$$\begin{pmatrix} 24 & 0 \\ 10 & 6 \end{pmatrix}.$$

Exercice 3.2.6.

Pour chacune des matrices de confusion suivantes, déterminer

- le nombre d'étiquettes ;
- la taille de l'échantillon de test ;
- le nombre d'observations de chaque étiquette dans l'échantillon de test (on pourra les numérotter à partir de 0) ;
- l'erreur de classification.

$$1. A = \begin{pmatrix} 10 & 2 & 1 \\ 3 & 8 & 4 \\ 2 & 7 & 1 \end{pmatrix}$$

$$2. B = \begin{pmatrix} 1 & 4 & 0 & 3 \\ 5 & 10 & 1 & 0 \\ 1 & 5 & 7 & 2 \\ 1 & 6 & 3 & 7 \end{pmatrix}$$

Exercice 3.2.7.

On considère une tâche de classification à trois étiquettes, et l'on construit deux classificateurs f_1, f_2 dessus, dont on donne les matrices de confusion, calculées sur un même échantillon de test.

$$M_1 = \begin{pmatrix} 10 & 2 & 3 \\ 0 & 7 & 1 \\ 2 & 1 & 5 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 11 & 1 & 3 \\ 2 & 2 & 4 \\ 1 & 0 & 7 \end{pmatrix}.$$

Quel classificateur choisissez-vous ?

Exercice 3.2.8.

On reprend l'exercice 2.1.7.

Calculer la matrice de confusion sur l'échantillon de test pour $K = 1$. Qu'en dites-vous ?