Algorithme des k plus proches voisins

Résumé

Comment un programme peut-il reconnaître une fleur, détecter un spam ou prédire si un patient est malade ? Une idée simple et intuitive : regarder les cas les plus **proches** déjà connus et s'en inspirer. C'est exactement ce que fait l'algorithme des k plus proches voisins (KNN) : pour classer un nouvel élément, il cherche les k éléments qui lui ressemblent le plus dans un jeu de données existant, puis il attribue la catégorie la plus fréquente parmi ces voisins. C'est l'un des algorithmes les plus simples de l'apprentissage automatique, et il illustre parfaitement l'idée que la proximité dans les données peut servir à prendre des décisions.

Prérequis : Les listes, Les fonctions.

① Mémo

Définition 1 (Principe de l'algorithme KNN)

L'algorithme des k **plus proches voisins** (*k-nearest neighbors*, ou KNN) est un algorithme de **classification** : étant donné un nouvel élément, il prédit sa catégorie en regardant les catégories des k éléments les plus proches dans un jeu de données existant.

Étapes :

1. Calculer la distance entre le nouvel élément et chaque élément du jeu de données.
2. Trier les éléments par distance croissante.
3. Sélectionner les k plus proches.
4. Déterminer la catégorie majoritaire parmi ces k voisins.

Définition 2 (La distance euclidienne)

Pour mesurer la proximité entre deux points, on utilise la **distance euclidienne**.

En dimension 2, entre les points $\mathrm{A}(x_1\ ; y_1)$ et $\mathrm{B}(x_2\ ; y_2)$:

$$d(\mathrm{A},\ \mathrm{B}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

En dimension n, entre les points $(a_1,\ a_2,\ \ldots,\ a_n)$ et $(b_1,\ b_2,\ \ldots,\ b_n)$:

$$d = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2}$$

Le choix de k

k trop petit (ex. $k = 1$) Le résultat dépend d'un seul voisin : très sensible au bruit et aux données aberrantes.

k trop grand On prend en compte des voisins trop éloignés, qui ne sont plus pertinents. La frontière entre les catégories devient floue.

Règle pratique Choisir k impair (pour éviter les égalités) et tester plusieurs valeurs.

Ordre de grandeur $k = \sqrt{n}$ (où n est le nombre de données) est un point de départ classique.

Normalisation des données

Si les caractéristiques n'ont pas la même échelle (par exemple une taille en mètres et un poids en kilogrammes), la caractéristique avec les plus grandes valeurs domine le calcul de distance.

Solution : normaliser chaque caractéristique pour que ses valeurs soient comprises entre 0 et 1 :

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Cas particulier : si tous les échantillons ont la même valeur pour une caractéristique ($x_{\max} = x_{\min}$), la formule donne une division par zéro. Cette caractéristique n'apporte aucune information pour distinguer les points : le plus propre est de la retirer du jeu de données. Si on la garde, on lui assigne une valeur neutre comme 0,5 (milieu de l'intervalle) plutôt que 0, qui biaiserait artificiellement vers la borne basse.

Avantages et limites

Avantages Simple à comprendre et à implémenter. Aucun entraînement préalable (algorithme paresseux). Fonctionne bien avec peu de données.

Limites Lent pour de grands jeux de données (il faut calculer toutes les distances à chaque prédiction). Sensible aux données non normalisées. Sensible au bruit si k est trop petit.

⚠ Pièges fréquents

- Oublier de normaliser les données : la caractéristique la plus grande écrase les autres.
- Prendre k pair : risque d'égalité entre deux catégories sans moyen de trancher.
- Confondre la distance euclidienne avec la distance de Manhattan ($|x_2-x_1|+|y_2-y_1|$).
- Oublier la racine carrée dans la distance (cela ne change pas le classement, mais fausse l'interprétation des valeurs).

⚠ Erreurs classiques

Erreur	Correction	Explication
Prédire avec $k = 1$	Utiliser $k \geqslant 3$ (impair)	Avec $k = 1$, une seule donnée aberrante suffit à fausser la prédiction.
Ne pas normaliser (taille en cm, poids en kg)	Normaliser entre 0 et 1	La taille (150-190) dominerait le poids (50-100) dans le calcul de distance.
Trier par distance décroissante	Trier par distance **croissante**	On veut les voisins les plus **proches**, donc les distances les plus petites.
Compter les votes sans gérer l'égalité	Choisir k impair ou ajouter une règle de départage	k impair évite l'égalité uniquement quand il n'y a que deux catégories. Avec trois catégories ou plus, un partage 1−1−1 reste possible avec $k = 3$.

② Exemples

Programme 1 · Distance euclidienne en Python

```python
import math

def distance(a, b):
    """Distance euclidienne entre deux points (listes de coordonnées)."""
    return math.sqrt(sum((ai - bi) ** 2 for ai, bi in zip(a, b)))

# En 2D
print(distance([1, 2], [4, 6]))  # 5.0

# En 3D
print(distance([0, 0, 0], [1, 1, 1]))  # 1.732...
```

Programme 2 · Algorithme KNN complet

```python
def knn(donnees, nouveau, k):
    """Prédit la catégorie du point 'nouveau'.
    donnees : liste de tuples (coordonnees, categorie)
    nouveau : liste de coordonnées
    k : nombre de voisins."""
    # Étape 1 : calculer les distances
    distances = []
    for coords, cat in donnees:
        d = distance(coords, nouveau)
        distances.append((d, cat))

    # Étape 2 : trier par distance croissante
    distances.sort()

    # Étape 3 : sélectionner les k plus proches
    voisins = distances[:k]

    # Étape 4 : vote majoritaire
    decompte = {}
    for d, cat in voisins:
        decompte[cat] = decompte.get(cat, 0) + 1

    # Renvoyer la catégorie la plus fréquente.
    # En cas d'égalité, max() renvoie la première clé rencontrée dans le
    # dictionnaire, donc le résultat dépend de l'ordre des voisins. Pour
    # un code robuste, il faudrait détecter l'égalité et appliquer une
    # règle de départage (par exemple, la catégorie du voisin le plus proche).
    return max(decompte, key=decompte.get)
```

Programme 3 · Exemple concret : classification de fleurs

```python
import math

def distance(a, b):
    return math.sqrt(sum((ai - bi) ** 2 for ai, bi in zip(a, b)))

def knn(donnees, nouveau, k):
    distances = []
    for coords, cat in donnees:
        d = distance(coords, nouveau)
        distances.append((d, cat))
    distances.sort()
    voisins = distances[:k]
    decompte = {}
    for d, cat in voisins:
        decompte[cat] = decompte.get(cat, 0) + 1
    return max(decompte, key=decompte.get)

# Données : (longueur pétale, largeur pétale) -> espèce
fleurs = [
    ([1.4, 0.2], "setosa"),
    ([1.3, 0.3], "setosa"),
    ([4.5, 1.5], "versicolor"),
    ([4.7, 1.4], "versicolor"),
    ([5.1, 1.8], "virginica"),
    ([5.9, 2.1], "virginica"),
]

# Prédire l'espèce d'une nouvelle fleur
nouvelle = [4.6, 1.4]
print(knn(fleurs, nouvelle, k=3))  # "versicolor"
```

Programme 4 · Normalisation min-max

```python
def normaliser(donnees):
    """Normalise chaque caractéristique entre 0 et 1."""
    n = len(donnees[0][0])  # nombre de caractéristiques
    mins = [min(d[0][i] for d in donnees) for i in range(n)]
    maxs = [max(d[0][i] for d in donnees) for i in range(n)]
    resultat = []
    for coords, cat in donnees:
        # Si maxs[i] == mins[i], la caractéristique est constante et
        # n'apporte aucune info. On assigne 0.5 (valeur neutre) plutôt
        # que de diviser par zéro. Mieux : retirer cette caractéristique.
        norm = [(coords[i] - mins[i]) / (maxs[i] - mins[i])
                if maxs[i] != mins[i] else 0.5
                for i in range(n)]
        resultat.append((norm, cat))
    return resultat, mins, maxs
```

③ Exercices

Exercice 1 *Calcul de distances* Soit les points A(1 ; 3), B(4 ; 7), C(2 ; 1) et D(5 ; 4).

1. Calculer à la main $d(A, B)$, $d(A, C)$, $d(A, D)$.
2. Classer ces points du plus proche au plus éloigné de A.
3. Vérifier les résultats en Python.

Exercice 2 *Classification à la main* On dispose du jeu de données suivant (taille en cm, poids en kg, catégorie) :

Taille (cm)	Poids (kg)	Catégorie
160	55	A
165	60	A
170	70	B
175	75	B
180	80	B
155	50	A

1. Un nouvel individu mesure 168 cm et pèse 65 kg. Calculer sa distance à chaque point du jeu de données.
2. Pour $k = 3$, quels sont les trois plus proches voisins ? Quelle catégorie prédit KNN ?
3. Expliquer pourquoi la normalisation ne changerait pas le résultat dans ce cas précis (les deux caractéristiques ont des échelles similaires).

Exercice 3 *Implémentation complète*

1. Écrire une fonction `distance(a, b)` qui calcule la distance euclidienne entre deux points (représentés comme des listes de coordonnées).
2. Écrire une fonction `knn(donnees, nouveau, k)` qui prédit la catégorie d'un nouveau point.
3. Tester avec le jeu de données de l'exercice précédent. Vérifier que pour $k = 3$, l'algorithme prédit la catégorie B pour le point $(168, 65)$.
4. Tester avec $k = 1$ et $k = 5$. Le résultat change-t-il ?

Exercice 4 *Impact de la normalisation* On dispose des données suivantes (revenu en euros, âge en années, catégorie) :

Revenu	Âge	Catégorie
25000	30	X
30000	25	X
80000	45	Y
75000	50	Y

1. Un nouvel individu a un revenu de 35 000 et un âge de 40. Sans normalisation, calculer la distance à chaque point et prédire avec $k = 3$.
2. Normaliser les données (min-max) et recalculer. La prédiction change-t-elle ?
3. Expliquer pourquoi la normalisation est indispensable ici.

Solutions des exercices

Corrigé de l'exercice 1.

1. Calculs :
   - $d(\text{A},\text{B}) = \sqrt{(4-1)^2+(7-3)^2} = \sqrt{9+16} = \sqrt{25} = 5$;
   - $d(\text{A},\text{C}) = \sqrt{(2-1)^2+(1-3)^2} = \sqrt{1+4} = \sqrt{5} \approx 2{,}24$;
   - $d(\text{A},\text{D}) = \sqrt{(5-1)^2+(4-3)^2} = \sqrt{16+1} = \sqrt{17} \approx 4{,}12$.
2. Du plus proche au plus éloigné : C ($\sqrt{5}$), D ($\sqrt{17}$), B (5).
3. Vérification :

```python
import math

def distance(a, b):
    return math.sqrt(sum((ai - bi) ** 2 for ai, bi in zip(a, b)))

A, B, C, D = [1,3], [4,7], [2,1], [5,4]
print(distance(A, B))  # 5.0
print(distance(A, C))  # 2.236...
print(distance(A, D))  # 4.123...
```

Corrigé de l'exercice 2.

1. Distances au point (168 ; 65) :
   - (160, 55) : $\sqrt{(168-160)^2+(65-55)^2} = \sqrt{64+100} = \sqrt{164} \approx 12{,}81$ (A) ;
   - (165, 60) : $\sqrt{9+25} = \sqrt{34} \approx 5{,}83$ (A) ;
   - (170, 70) : $\sqrt{4+25} = \sqrt{29} \approx 5{,}39$ (B) ;
   - (175, 75) : $\sqrt{49+100} = \sqrt{149} \approx 12{,}21$ (B) ;
   - (180, 80) : $\sqrt{144+225} = \sqrt{369} \approx 19{,}21$ (B) ;
   - (155, 50) : $\sqrt{169+225} = \sqrt{394} \approx 19{,}85$ (A).
2. Tri par distance croissante : (170, 70) à 5,39 (B), (165, 60) à 5,83 (A), (175, 75) à 12,21 (B). Les trois plus proches voisins sont : B, A, B. Vote majoritaire : **B** (deux voix contre une).
3. La taille varie de 155 à 180 (amplitude 25) et le poids de 50 à 80 (amplitude 30). Ces amplitudes sont du même ordre de grandeur, donc aucune caractéristique ne domine le calcul de distance. La normalisation ne modifierait pas le classement.

Corrigé de l'exercice 3.

```python
import math

def distance(a, b):
    return math.sqrt(sum((ai - bi) ** 2 for ai, bi in zip(a, b)))

def knn(donnees, nouveau, k):
    distances = []
    for coords, cat in donnees:
        d = distance(coords, nouveau)
        distances.append((d, cat))
    distances.sort()
    voisins = distances[:k]
    decompte = {}
    for d, cat in voisins:
        decompte[cat] = decompte.get(cat, 0) + 1
    return max(decompte, key=decompte.get)

# Jeu de données
donnees = [
    ([160, 55], "A"), ([165, 60], "A"),
    ([170, 70], "B"), ([175, 75], "B"),
    ([180, 80], "B"), ([155, 50], "A"),
]
nouveau = [168, 65]

print(knn(donnees, nouveau, k=3))  # "B"
print(knn(donnees, nouveau, k=1))  # "B" (seul voisin : (170,70))
print(knn(donnees, nouveau, k=5))  # "B" (3 B, 2 A)
```

Analyse : avec $k = 1$, le plus proche voisin est $(170, 70)$ de catégorie B. Avec $k = 5$, on inclut trois points B et deux points A. Sur ce jeu de données, la prédiction reste donc B pour $k = 1$, $k = 3$ et $k = 5$: le point $(168, 65)$ se trouve dans une zone dominée par la catégorie B. Ce n'est pas une généralité : sur d'autres jeux de données, changer k peut faire basculer la prédiction. Le choix de k influence directement le résultat : c'est un **hyperparamètre** qu'il faut ajuster.

Corrigé de l'exercice 4.

1. Sans normalisation, distances au point $(35000, 40)$:
   - $(25000, 30)$: $\sqrt{(10000)^2 + (10)^2} = \sqrt{10^8 + 100} \approx 10\,000$ (X) ;
   - $(30000, 25)$: $\sqrt{(5000)^2 + (15)^2} \approx 5\,000$ (X) ;
   - $(80000, 45)$: $\sqrt{(45000)^2 + (5)^2} \approx 45\,000$ (Y) ;
   - $(75000, 50)$: $\sqrt{(40000)^2 + (10)^2} \approx 40\,000$ (Y).

   Les trois plus proches : $(30000, 25)$, $(25000, 30)$, $(75000, 50)$ $\to$ 2 X, 1 Y $\to$ prédiction **X**.
2. Normalisation : revenu varie de 25 000 à 80 000 (amplitude 55 000), âge varie de 25 à 50 (amplitude 25).
   - $(25000, 30) \to (0{,}00\ ; 0{,}20)$, $(30000, 25) \to (0{,}09\ ; 0{,}00)$;
   - $(80000, 45) \to (1{,}00\ ; 0{,}80)$, $(75000, 50) \to (0{,}91\ ; 1{,}00)$;
   - nouveau : $(35000, 40) \to (0{,}18\ ; 0{,}60)$.

   Distances normalisées :

- $(0{,}00\ ;0{,}20)$: $\sqrt{0{,}18^2+0{,}40^2} \approx 0{,}44$ (X) ;
- $(0{,}09\ ;0{,}00)$: $\sqrt{0{,}09^2+0{,}60^2} \approx 0{,}61$ (X) ;
- $(1{,}00\ ;0{,}80)$: $\sqrt{0{,}82^2+0{,}20^2} \approx 0{,}84$ (Y) ;
- $(0{,}91\ ;1{,}00)$: $\sqrt{0{,}73^2+0{,}40^2} \approx 0{,}83$ (Y).

Les trois plus proches : X $(0{,}44)$, X $(0{,}61)$, Y $(0{,}83)$ $\rightarrow$ prédiction **X**. Même résultat ici, mais les distances sont bien plus équilibrées.

3. La normalisation est indispensable car le revenu (valeurs de l'ordre de 10^4) écrase l'âge (valeurs de l'ordre de 10^1) dans le calcul de distance. Sans normalisation, l'âge ne pèse quasiment rien : c'est comme si on classait uniquement par revenu.