k plus proches voisins (k-PPV ou k-NN)

1. Définition et idée générale

L'algorithme des **k plus proches voisins** (k-PPV ou k-NN, pour k-*Nearest Neighbours* en anglais) propose une classe **possible** pour un « objet inconnu » en cherchant la classe majoritaire parmi les `k` objets « les plus proches » (au sens d'une distance calculée à partir de caractéristiques numériques).

On dit parfois que cet algorithme est un *classifieur*.

> ⚠️ **Attention !**
>
> Le résultat dépend de la valeur choisie pour `k` : différentes valeurs de `k` peuvent amener à des propositions de classification différentes.
>
> > 💭 **Remarque : autre dépendance (hors programme)** ▼
> >
> > Le résultat dépend aussi du choix de la distance (car il existe différentes distances… mais cette remarque est largement hors programme du secondaire). Vous connaissez la distance habituelle dans le plan, appelée *distance* ***euclidienne***. Mais en informatique (par exemple), il est courant de rencontrer la **distance de *Manhattan*** (https://fr.wikipedia.org/wiki/Distance_de_Manhattan).

1.1. Distance usuelle en 2 dimensions

```
def distance(xA, yA, xB, yB):
    return ((xB - xA)**2 + (yB - yA)**2)**0.5    # puissance 0.5 correspond à la racine carré
```

> 💭 **Remarque : vocabulaire** ▼
>
> Cette distance est appelée *distance* ***euclidienne***.

> 💭 **Remarque : inutilité du calcul de la racine carrée** ▼
>
> La fonction $x \longmapsto \sqrt{x}$ est une fonction *croissante*, donc on peut se contenter de calculer le *carré* de la distance classique (un calcul en moins peut avoir un impact favorable sur le temps d'exécution, lorsqu'il y a un grand nombre de valeurs étiquetées — et cela sans rien changer au résultat). La fonction ci-dessous convient donc mieux, en fait (mais ça *n'*est *pas* une distance au sens mathématique du terme) :

```python
def distance2(xA, yA, xB, yB):
    return (xB - xA)**2 + (yB - yA)**2
```

1.2. Distance usuelle en dimensions supérieures

Si on dispose de 3 paramètres numériques, on utilisera la fonction suivante :

```python
def distance2(xA, yA, zA, xB, yB, zB):
    return (xB - xA)**2 + (yB - yA)**2 + (zB - zA)**2
```

💭 **Remarque : pour aller plus loin ▼**

Vous pouvez constater qu'une fonction qui calculerait la distance en dimension 17 (avec 17 paramètres chiffrés connus pour chaque objet déjà étiqueté) serait pénible à écrire en suivant le modèle précédent, où les paramètres sont **tous** explicitement mentionnés. Désormais, on note `A` et `B` deux tuples (ou deux listes) de n éléments (dans l'exemple ci-dessous, n vaut 4). On peut calculer la distance entre `A` et `B` dans un espace de dimension n grâce au code suivant :

```python
def distance2(A, B):
    return sum([(A[i]-B[i])**2 for i in range(len(A))])

A = (1, 2, 3, 4)
B = (5, 6, 7, 8)
distance2(A, B)                # renvoie 64
```

💭 **Remarque : vous voulez aller *encore* plus loin ? ▼**

Voici un code « plus pythonique » :

```python
def distance2(A, B):
    return sum((a - b)**2 for a, b in zip(A, B))
```

L'instruction `zip(A, B)` regroupe les éléments deux à deux : `(A[0], B[0])`, `(A[1], B[1])`, etc., et *l'unpacking* `a, b` décompose chaque paire directement dans la boucle. **Cette seconde remarque est largement *hors programme* de NSI !**

2. Illustration

Pour reconnaître un fruit *inconnu*, on peut comparer sa taille, sa masse et sa couleur à ceux de fruits *déjà connus* (on dit que ces fruits sont *étiquetés*). Les fruits les plus proches (en termes de taille, masse et couleur — des données

numériques, ou qu'on peut convertir en valeurs numériques) « votent » pour la classe finale : la variété la plus fréquente parmi les `k` fruits connus les plus proches servira de prédiction pour la classe du fruit inconnu.

3. Principe

L'idée est simple :

1. on calcule les distances entre l'objet inconnu et **tous** les objets connus ;
2. on garde les `k` plus petites distances ;
3. parmi les `k` objets correspondants à ces distances, on regarde si une classe est majoritaire ;
4. si oui, on *prédit* que l'objet inconnu appartient à cette classe.

💭 **Remarque : et s'il n'y a pas de classe majoritaire ?** ▼

Cela peut arriver, même si on prend la précaution élémentaire de prendre une valeur *impaire* pour `k` (3, 5, 7…). La situation peut se présenter dès qu'il y a plus de deux catégories possibles. Par exemple, quand on cherche à classifier un fruit inconnu, dans l'hypothèse où on prend `k=5` et qu'on dispose d'informations sur des pommes, des poires et des coings (des fruits parfois difficiles à distinguer), on peut avoir dans les 5 plus proches voisins deux poires et deux coings (et une pomme, donc). Comment conclure alors ? On peut soit modifier `k` en espérant trouver alors une classe majoritaire, soit choisir une règle d'arbitrage (par exemple un tirage au sort parmi les classes à égalité).

4. Points d'attention complémentaires

- Avec `k = 1`, la classification sera très locale : elle sera très sensible aux valeurs aberrantes.
- Une valeur trop grande pour `k` peut compliquer la classification.
- Le choix de la distance est fondamental.
- k-NN ne correspond à aucune règle préétablie : c'est juste une comparaison à des exemples déjà connus.
- Il est pertinent, lorsqu'on a un gros volume de données étiquetées, d'éviter un tri en sélectionnant les `k` distances les plus faibles au fur et à mesure qu'on les calcule.

5. Liens avec d'autres notions

- La **complexité** est importante, car il faut souvent comparer l'objet à *beaucoup* d'exemples.
- Les **tris** peuvent intervenir pour sélectionner les voisins les plus proches (mais trier n'est pas obligatoire, on peut sélectionner les `k` meilleures distances au fur et à mesure).