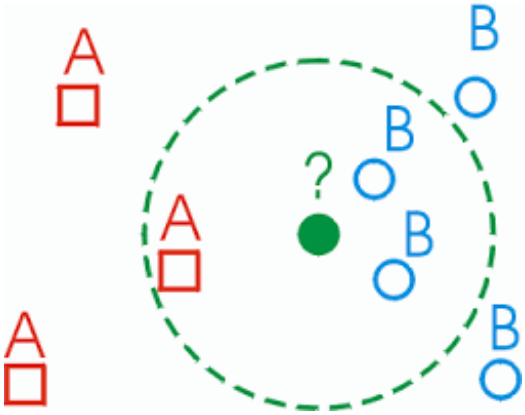


ALGORITHME DES k PLUS PROCHES VOISINS (KNN)

“

Celui qui marche avec des hommes sages sera sage, mais le compagnon d’imbéciles souffrira.

”

(Proverbe)

Numérique et Sciences Informatiques
1ère

Support de cours :
Jean-Christophe BONNEFOY

Objectifs :

- Savoir écrire un algorithme qui prédit la classe d’un élément en fonction de la classe majoritaire de ses k plus proches voisins.

1. Contexte

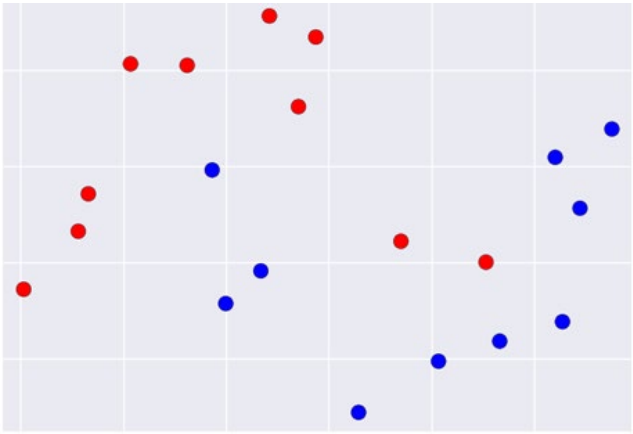
L'algorithme des k plus proches voisins s'écrit souvent KNN de l'anglais *K Nearest Neighboors*. K étant un nombre entier positif généralement petit. **Le principe de ce modèle consiste en effet à choisir les** k **données les plus proches du point étudié afin d'en prédire sa valeur**.

Il s'agit d'un algorithme de **machine learning** ou « apprentissage machine » qui est essentiel dans le milieu de l'intelligence artificielle.

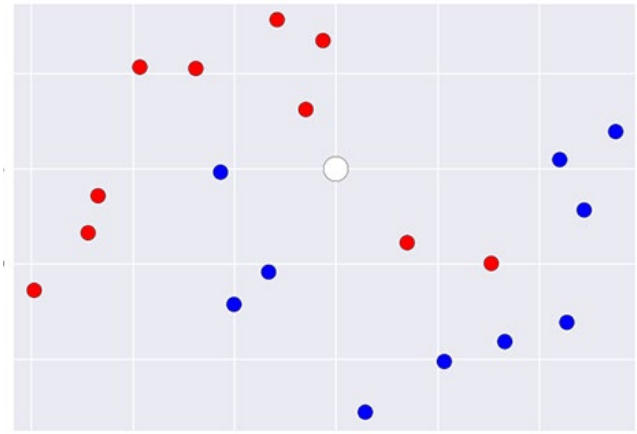
Le principe peut être résumé par « Dis-moi qui sont tes amis, et je te dirais qui tu es ». Ce genre d'algorithme est utilisé par exemple pour prédire le comportement d'une personne en s'intéressant à son milieu. Les géants de la vente en ligne comme Amazon, Netflix, … l'utilise afin de prévoir si vous seriez intéressé ou non par un produit, un film, … En effet, en utilisant vos données (âge, derniers achats, …) et en les comparant à celle d'un client ayant acheté un article, ils peuvent prédire si vous seriez intéressé ou non par cet article.

2. Fonctionnement du modèle KNN

Prenons un exemple visuel afin de comprendre le fonctionnement de cet algorithme. Nous avons un jeu de données constitué d'éléments appartenant à deux classes différentes : une classe rouge et une classe bleue.

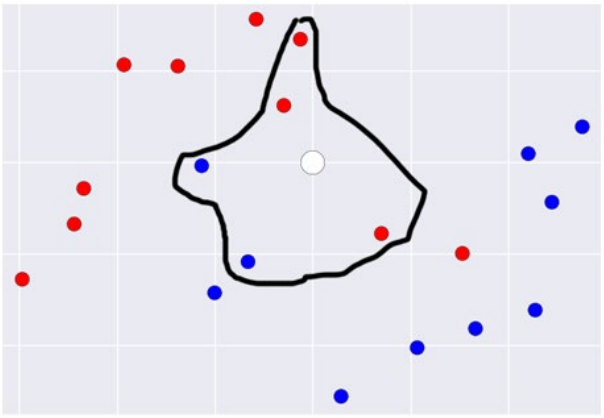


Essayons de prédire la classe d'une nouvelle entrée (représentée en blanc).



On va simplement regarder les **k** voisins les plus proches de ce point et regarder quelle classe constitue la majorité de ces points, afin d'en déduire la classe du nouveau point.

Par exemple, ici, si on utilise le 5-NN (c'est-à-dire que k = 5), on peut prédire que la nouvelle donnée appartient à la classe **rouge** puisqu'elle possède 3 données rouges et 2 données bleues dans son entourage.



Remarque : On a utilisé ici comme mesure de similarité la distance euclidienne. On peut bien sûr utiliser d'autres normes.

Cet algorithme est qualifié de paresseux (*Lazy Learning*) car il n'apprend rien pendant la phase d'entraînement. Pour prédire la classe d'une nouvelle donnée d'entrée, il va chercher ses K voisins les plus proches (en utilisant la distance euclidienne, ou autres) et choisira la classe des voisins majoritaires.

Pour appliquer cette méthode, les étapes à suivre sont les suivantes :

- On fixe le nombre de voisins k.
- On détecte les k-voisins les plus proches des nouvelles données d'entrée que l'on veut classer.
- On attribue les classes correspondantes par vote majoritaire.

3. Méthodologie

Dans le cadre du programme de NSI, on ne s'intéressera qu'à la prédiction de la classe d'un nouvel élément (dont la position est connue) faisant partie d'un ensemble de points dont les coordonnées et la classe sont connues.

Pour ce nouvel élément, il faudra donc :

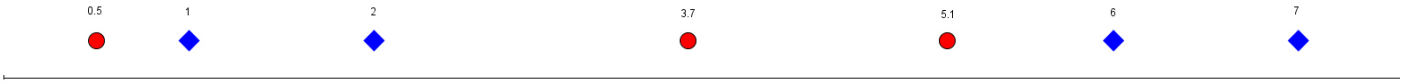
- Évaluer la distance qui le sépare de chacun des autres éléments de l'ensemble.
- Stocker ces valeurs de distance en mémoire.
- Trier ces valeurs de distances suivant l'ordre croissant.
- Restreindre cette liste de valeurs aux k premiers éléments.
- Assigner une classe à ce nouvel élément en fonction de la majorité des classes représentées parmi les k plus proches voisins.

Dans la pratique, il convient de mettre en œuvre au minimum 3 fonctions :

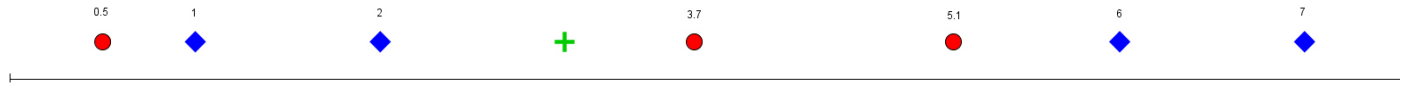
- Une fonction ***Distance*** pour le calcul de la distance entre 2 points de coordonnées connues.
- Une fonction ***KVoisins*** qui détermine les k plus proches voisins d'un nouvel élément.
- Une fonction ***PredireLaClasse*** qui détermine le résultat majoritaire des classes d'appartenance des k plus proches voisins et assigne la classe du nouvel élément à cette classe majoritaire.

4. Application sur des éléments disposés sur une seule dimension (1D)

On dispose des éléments suivants (d'abscisse connue) répartis en 2 classes : Cercles et Losanges



On cherche à prédire la classe d'un nouvel élément d'abscisse 3 (représenté ci-dessous en vert).



4.1 Définition des données

Il existe bien évidemment plusieurs façons de définir nos éléments : listes, tuples, dictionnaires… On pourra dans la suite de ce cours définir nos données avec des listes.

Par exemple :

> *Abscisses = [0.5, 1.0, 2.0, 3.7, 5.1, 6.0, 7.0]*
> *Classes = ['C', 'L', 'L', 'C', 'C', 'L', 'L']*

4.2 La fonction *Distance*

Dans un « repère » à une seule dimension, la distance d entre 2 éléments d'abscisse réelle x_1 et x_2 est définie par :

$$d = |x_1 - x_2|$$

On a donc :

```
Fonction Distance(abs1, abs2) :
    Retourner Valeur absolue de (abs1-abs2)
```

4.3 La fonction *Kvoisins*

Il nous faut écrire la fonction *Kvoisins* prenant en paramètres la liste des abscisses des éléments *(L)*, le nombre de voisins *(k)* et le nouvel élément à classifier *(nouvel_el)*. Cette fonction doit nous retourner une liste des indices de la liste L des k plus proches voisins. On peut ainsi proposer :

```
Fonction Kvoisins(L, k, nouvel_el) :
    ListeDistIndice = []
    Pour i allant de 0 à Longeur(L) -1 par pas de 1 :
        D = Distance(nouvel_el, i)
        Ajouter[D,i] à ListeDistIndice
    Trier ListeDistIndice selon D
    Voisins = []
    Pour i allant de 0 à k -1 par pas de 1 :
        Ajouter ListeDistIndice[i][1] à Voisins
    Retourner Voisins
```

4.4 La fonction *PredireLaClasse*

Enfin, il nous reste à écrire la fonction *PredireLaClasse* qui « calcule » le résultat majoritaire des classes d'appartenance des k plus proches voisins et fixe ainsi la classe du nouvel élément. Cette fonction prend donc comme paramètres : La liste des indices des k plus proches voisins (retournée par la fonction *Kvoisins*), la liste des classes. Elle retournera la classe majoritaire.

On peut donc proposer :

```
Fonction PredireLaClasse(LindiceV, Classe) :
    Dico_decompte = {}
    Pour chaque élément v de LindiceV :
        Si Classe[v] est présent dans Dico_decompte
            On incrémente de 1 la valeur de Dico_decompte de clé 'Classe[v]'
        Sinon
            On crée une entrée dans Dico_decompte de clé 'Classe[v]' et de valeur 1
    val_max=0
    classe_max = ""
    Pour chaque entrée de Dico_decompte :
        Si la valeur de l'entrée > val_max :
            Val_max = valeur de l'entrée
            Classe_max = clé de l'entrée
    Retourner classe_max
```

4.5 Le programme en entier

```
Abscisses = [0.5, 1.0, 2.0, 3.7, 5.1, 6.0, 7.0]
Classes = ['C', 'L', 'L', 'C', 'C', 'L', 'L']
nouvel_element = 3.0
k = 3
lst_voisins = Kvoisins(Abscisses, k, nouvel_element)
Afficher("La classe d'appartenance prédite de l'élément d'abscisse", nouvel_element, "est")
Afficher(PredireLaClasse(lst_voisins, Classes ))
```

Remarque :

- Pour k = 3, on obtient la classification 'L' pour le nouvel élément.
- Pour k = 5, on obtient la classification 'C' pour le nouvel élément