Partie 1

Introduction à l'algorithme KNN et son utilisation avec SQL

1.1 1. Principe de l'algorithme KNN (K-Nearest Neighbors)

L'algorithme **KNN** (K plus proches voisins) est une méthode de classification ou de recherche basée sur la **similarité**. Le principe est simple :

> Pour classifier ou analyser un objet, on cherche les K objets les plus proches dans notre base de données, puis on utilise leurs caractéristiques pour prendre une décision.

1.1.1 Exemple concret : Classification d'exoplanètes

Imaginons que nous voulons déterminer si une nouvelle planète découverte est plutôt de type "terrestre" (comme la Terre) ou "gazeuse" (comme Jupiter).

Démarche KNN : 1. On calcule la distance entre cette nouvelle planète et toutes les planètes connues 2. On sélectionne les K planètes les plus proches (par exemple K=5) 3. On regarde à quel type appartiennent ces 5 voisins (si les objets appartiennent à un certain type) 4. On attribue le type majoritaire à notre nouvelle planète

1.2 2. Calcul de la distance

La distance utilisée est généralement la **distance euclidienne** dans un espace à plusieurs dimensions.

Pour une exoplanète caractérisée par sa **masse** (mass) et son **rayon** (radius), la distance à Jupiter est :

```
distance = √[(mass - mass_Jupiter)² + (radius - radius_Jupiter)²]
```

1.2.1 Visualisation

Sur un graphique avec la masse en abscisse et le rayon en ordonnée : - Chaque planète est un point - Jupiter est notre point de référence - La distance KNN correspond au rayon d'un cercle centré sur Jupiter - Les planètes à l'intérieur du cercle sont les "voisins proches"

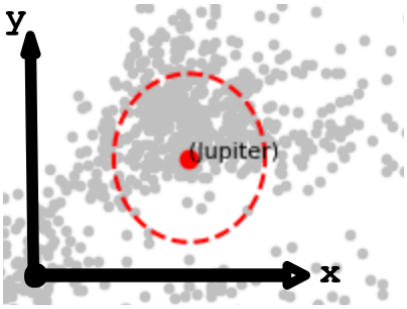

FIGURE 1 – Jupiter dans un diagramme en nuage de points masse-rayon

1.3 3. Implémentation en SQL

1.3.1 A. Création de la fonction de distance

SQLite ne possède pas nativement la fonction `power()` ni toutes les fonctions mathématiques. On doit donc les créer en Python :

```python
import sqlite3
import math

# Connexion à la base de données
conn = sqlite3.connect('exoplanetes.db')

# Création de la fonction power
conn.create_function("power", 2, lambda x, y: x**y if x != None and y
    != None else None)

# Création de la fonction sqrt
conn.create_function("sqrt", 1, lambda x: math.sqrt(x) if x != None and
     x >= 0 else None)
```

1.3.2 B. Calcul de la distance dans une requête SQL

Une fois les fonctions créées, on peut calculer directement la distance dans nos requêtes :

```sql
SELECT
    name,
    mass,
    radius,
    sqrt(power(mass - 1.0, 2) + power(radius - 1.0, 2)) as
   distance_jupiter
FROM exoplanets
WHERE mass IS NOT NULL
  AND radius IS NOT NULL
ORDER BY distance_jupiter
LIMIT 5;
```

QUESTION 1 : Que retourne la requête sql ci-dessus ?

1.3.3 C. Utilisation d'une fonction personnalisée

Pour simplifier et rendre le code plus lisible, on peut créer une fonction SQL dédiée :

```python
def distance_jupiter_sql(mass, radius):
    """Calcule la distance euclidienne à Jupiter"""
    if mass is None or radius is None:
        return None

    mass_J = 1.0    # Masse de Jupiter (en masses joviennes)
    radius_J = 1.0  # Rayon de Jupiter (en rayons joviens)

    import math
    distance = math.sqrt((mass - mass_J)**2 + (radius - radius_J)**2)

    return distance

# Enregistrement dans SQLite
```

```
conn.create_function("distance_jupiter", 2, distance_jupiter_sql)
```

Utilisation simplifiée :

```
SELECT
    name,
    mass,
    radius,
    distance_jupiter(mass, radius) as distance_jupiter
FROM exoplanets
WHERE distance_jupiter(mass, radius) IS NOT NULL
ORDER BY distance_jupiter
LIMIT 5;
```

1.3.4 D. Mise à jour de la base avec les distances

On peut aussi ajouter une colonne dans la table pour stocker les distances :

```
# Ajout de la colonne
cursor.execute("ALTER TABLE exoplanets ADD COLUMN distance_jupiter REAL
    ")

# Calcul et mise à jour
cursor.execute("""
    UPDATE exoplanets
    SET distance_jupiter = distance_jupiter(mass, radius)
""")

conn.commit()
```

1.4 4. Exercice pratique

> QUESTION 2 : Comment trouver les 10 exoplanètes les plus similaires à la Terre ? Compléter la série d'instruction ci-dessous

Données de référence

Important : Dans notre base de données, les masses et rayons sont exprimés **relativement à Jupiter**.

- Masse de la Terre : **0.00315** masses joviennes (la Terre est ~318 fois moins massive que Jupiter)
- Rayon de la Terre : **0.0892** rayons joviens (le rayon de la Terre est ~11.2 fois plus petit que celui de Jupiter)

Étape 1 : Créer la fonction `distance_terre`

En vous inspirant de la fonction `distance_jupiter_sql` fournie ci-dessus, complétez le code suivant :

```
def distance_terre_sql(mass, radius):
    """Calcule la distance euclidienne à la Terre"""
    if mass is None or radius is None:
        return None

```

```
    mass_T = ______    # À compléter : masse de la Terre en masses joviennes
    radius_T = ______  # À compléter : rayon de la Terre en rayons joviens

    import math
    distance = math.sqrt((mass - ______)**2 + (radius - ______)**2)

    return distance

# Enregistrement dans SQLite
conn.create_function("______", 2, distance_terre_sql)
```

Étape 2 : Requête SQL

Complétez la requête SQL pour obtenir les 10 planètes les plus proches de la Terre :

```
SELECT
    name,
    mass,
    radius,
    ______(mass, radius) as distance_terre
FROM exoplanets
WHERE ___________(mass, radius) IS NOT NULL
ORDER BY ______
LIMIT ______;
```

QUESTION 3 : Compléter le script pour mettre à jour de la base.

Ajoutez une colonne `distance_terre` dans la table et calculez les distances :

```
# Ajout de la colonne
cursor.execute("ALTER TABLE exoplanets ADD COLUMN ___________ REAL")

# Calcul et mise à jour
cursor.execute("""
    UPDATE exoplanets
    SET ___________ = ___________(mass, radius)
""")

conn.commit()
```

1.5 5. Visualisation

La fonction `coordonnees_cercle_xlog_y` retourne les listes x,y de coordonnées des points du cercle centré sur X,Y, de rayon r. Cette fonction a pour paramètres X, Y, r.

Compléter le code de visualisation pour afficher la Terre et son cercle de voisinage (rayon = 0.3). Le point représentant la Terre sera de couleur bleue :

```
# Données Terre (en unités joviennes)
mass_T = 0.00315
radius_T = 0.0892
```

```
4
5  # Position de la Terre sur le graphique
6  plt.plot(mass_T, radius_T, marker='o', color='______')
7  axes.text(mass_T, radius_T, f"(Terre)", fontsize=8)
8
9  # Cercle autour de la Terre (à compléter)
10 x_circle_T, y_circle_T = coordonnees_cercle_xlog_y(______, ______,
       ______)
11 plt.plot(x_circle_T, y_circle_T, color='gold', linewidth=1.5)
```

QUESTION 4 :

1. Pourquoi est-il important d'utiliser les mêmes unités (masses et rayons joviens) pour toutes les planètes ?
2. Observez les valeurs de masse et rayon de la Terre par rapport à Jupiter. Que remarquez-vous ? Ces valeurs sont-elles du même ordre de grandeur ?
3. Les 10 planètes les plus proches de la Terre sont-elles toutes de type "terrestre" ?

Note : Dans le TP à venir, nous verrons comment adapter cette distance lorsque les données ont des échelles très différentes (par exemple avec des échelles logarithmiques).

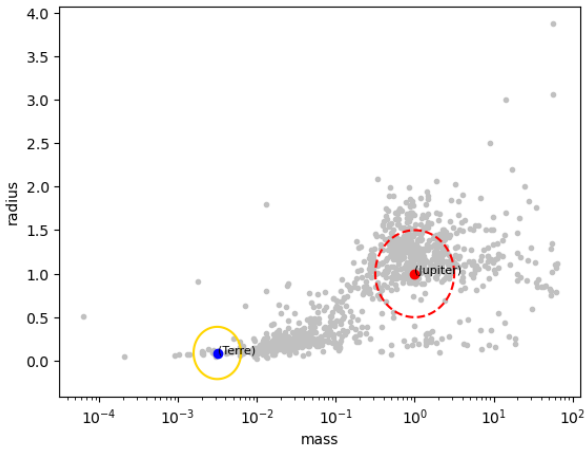

FIGURE 2 – représentation graphique avec axe des abscisses logarithmique