SAE Classification, rendu de développement efficace

Implémentation de l'algorithme K-NN

Cette implémentation se trouve dans la fonction `getNearestDatas` de la classe `DataManager`.

L'implémentation de K-NN (K-Nearest Neighbors) utilise une `PriorityQueue` pour identifier efficacement les `nbVoisin` (K) plus proches voisins d'une donnée cible. Voici les points clés de cette approche :

1. **PriorityQueue et MaxHeap** : L'emploi d'un `maxHeap` permet de maintenir les distances des K plus proches voisins de manière optimisée, avec des opérations d'insertion et de recherche en O(log K), contrairement à un tri complet qui aurait une complexité en O(N log N).
2. **Calcul des distances** : Pour chaque donnée dans `dataList`, la distance à la donnée cible est calculée via `Data.distance(data, d, distanceSouhaitee)`, ce qui permet de garder uniquement les K plus proches voisins et de réduire le nombre de comparaisons nécessaires.
3. **Gestion du MaxHeap** :
   - Si le `maxHeap` a une taille inférieure à K, la nouvelle donnée est ajoutée.
   - Si le `maxHeap` est plein et qu'une donnée a une distance inférieure à celle du maximum, elle remplace cet élément, garantissant ainsi la conservation des K voisins les plus proches.
4. **Extraction des résultats** : Les données des K plus proches voisins sont extraites du `maxHeap` et retournées dans une liste.

Efficacité sur grands volumes de données

- **Complexité temporelle** : La complexité est O(N log K), optimisée pour le nombre de voisins à retrouver, plutôt qu'O(N log N).
- **Mémoire** : Le `maxHeap` fixe de taille K permet une gestion efficace de la mémoire, même avec des `dataList` volumineuses, réduisant l'empreinte mémoire de l'algorithme. (Une PriorityQueue est par default de taille 11, elle est dynamique et est augmentée si nécessaire)
- **Scalabilité** : Cette méthode est capable de traiter des ensembles de données croissants, car le temps de traitement dépend principalement du nombre de voisins et non de la taille totale des données.

Calcul de la Distance

La classe `Data` contient une méthode `distance` qui permet de calculer la distance entre deux objets `Data`. Cette méthode utilise la distance souhaitée pour effectuer

le calcul.

Distances Disponibles

Les distances disponibles sont :

- **Distance de Manhattan** : La distance de Manhattan est la somme des différences absolues entre les coordonnées des points.
- **Distance Euclidienne** : La distance euclidienne est la racine carrée de la somme des carrés des différences entre les coordonnées des points.
- **Distance de Manhattan Normalisée** : La distance de Manhattan normalisée est la somme des différences absolues entre les coordonnées normalisées des points.
- **Distance Euclidienne Normalisée** : La distance euclidienne normalisée est la racine carrée de la somme des carrés des différences entre les coordonnées normalisées des points.

Ces distances permettent de comparer les objets `Data` en fonction de leurs attributs, en prenant en compte les valeurs numériques et les attributs sans ordre spécifique.

Traitement des Attributs Sans Ordre Spécifique

Lors du calcul de la distance entre deux objets `Data`, les attributs sans ordre spécifique sont traités différemment. Voici comment cela fonctionne :

- Si les attributs sont égaux, la distance est nulle.
- Sinon, un attribut est noté 1 et l'autre 0, ce qui permet de ne prendre en compte que l'égalité des attributs dans le calcul de la distance.

L'ordre est déterminé par la présence de l'annotation `@HasNoOrder` sur l'attribut lors de la déclaration de la classe.

Traitement de la normalisation

La normalisation est réalisée par les implémentations de `Distance` qui en ont besoin. Ces implémentations normalisent les valeurs des attributs avant de calculer la distance.

Méthode de Normalisation

La normalisation est effectuée en suivant les étapes suivantes :

1. Soustraction de la valeur minimale de chaque attribut.
2. Division par la plage de valeurs (max - min) de l'attribut.

Optimisation de la Normalisation

Pour accélérer le traitement, nous stockons le minimum et le maximum de chaque attribut dans la map `minMax` de la classe `DataManager`. Cela permet à la normalisation d'être exécutée en O(1) pour chaque attribut au lieu de recalculer les valeurs minimales et maximales à chaque fois O(N).

Classification des Données

La classification des données est réalisée par la méthode `categorizeData` de la classe `DataManager`. Cette méthode utilise l'algorithme K-NN pour classer une donnée cible en fonction de ses voisins les plus proches.

1. **Récupération des K plus proches voisins** : Les K plus proches voisins de la donnée cible sont extraits à l'aide de l'algorithme K-NN.
2. **Détermination de la classe majoritaire** : La classe majoritaire parmi les K voisins est déterminée.

Nous stockons un compteur pour chaque classe possible, et la classe majoritaire est celle avec le plus grand nombre de voisins.

Robustesse de l'Algorithme

Pour évaluer la robustesse de l'algorithme, nous avons mis en œuvre deux approches :

1. Robustesse Simple

L'utilisateur transmet un fichier contenant des données de test. L'algorithme K-NN est appliqué sur ces données, et les classes prédites sont comparées aux classes réelles pour évaluer la précision de l'algorithme. Le pourcentage de précision est calculé en fonction du nombre de classes prédites correctement.

2. Validation Croisée

L'utilisateur utilise la fonction de validation croisée, qui sépare les données actuelles en 10 sous-ensembles. À chaque itération, une sous-partie est testée tandis que les neuf autres servent à prédire les classes. À la fin des 10 itérations, nous calculons la moyenne des pourcentages de réussite.

Conditions de Test

Pour chaque approche, l'algorithme teste tous les N inférieurs à la moitié de la taille de la liste de données, car il n'est pas pertinent de tester pour un N trop grand. Le meilleur N est alors appliqué à toutes les futures classifications.

Des fichiers de resulats de Robustesse et de Validation Croisée sont disponibles à la racine du projet. Vous pouvez en visualiser d'autre dans le terminal en lancant

des tests de robustesse ou de validation croisée depuis l'inteface graphique.

Fonctionnement

Je vous renvoie à la section Fonctionnement de README.md pour plus de détails sur le fonctionnement de l'application. Vous y trouverez des informations notamment sur comment `Data` gere les types de données.