Formation: M2 SDTS

Module: Deep Learning

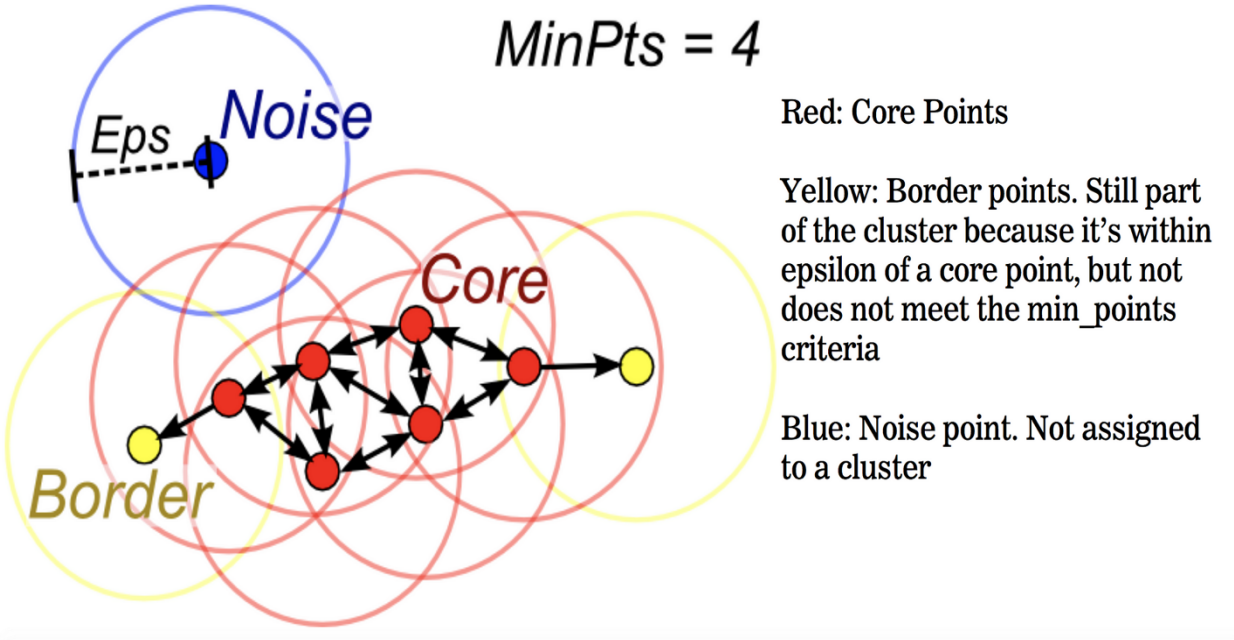
TP2: Implémentation de l'Algorithme DBSCAN en Python et Comparaison avec scikit-learn

Introduction

Dans ce TP, vous allez implémenter l'algorithme de clustering basé sur la densité DBSCAN en Python, sans utiliser d'implémentations préexistantes. Vous appliquerez ensuite votre implémentation sur trois jeux de données : Iris, Wine et Wholesale Customers, puis vous comparerez vos résultats avec ceux obtenus à l'aide de l'implémentation de scikit-learn.

Objectifs

- Comprendre en profondeur le fonctionnement de l'algorithme DBSCAN.
- Implémenter DBSCAN en Python à partir de zéro.
- Appliquer votre implémentation sur les jeux de données fournis.
- Comparer vos résultats avec ceux de scikit-learn.
- Analyser les performances et la précision de votre implémentation.



Étapes à Suivre

1. Comprendre l'Algorithme DBSCAN

Avant de commencer l'implémentation, assurez-vous de bien comprendre les concepts clés de DBSCAN :

- **ε (epsilon)** : distance maximale entre deux points pour qu'ils soient considérés comme voisins.
- **min_samples** : nombre minimum de points requis pour former un cluster (y compris le point central).
- **Point cœur (Core Point)** : point ayant au moins min_samples points dans son voisinage (dans un rayon ε).
- **Point frontière (Border Point)** : point qui n'est pas un point cœur mais qui est accessible à partir d'un point cœur.
- **Bruit (Noise Point)** : point qui n'est ni un point cœur ni un point frontière.

2. Implémenter DBSCAN en Python

a. Préparation

- Créez un nouveau fichier Python ou un notebook Jupyter.
- Importez les bibliothèques nécessaires :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

b. Définition de la Classe DBSCAN

Commencez par définir une classe DBSCAN avec les attributs essentiels :

```
class DBSCAN:
  def __init__(self, eps=0.5, min_samples=5):
    """
    Initialise l'algorithme DBSCAN avec les paramètres donnés.

    Paramètres:
```

```
    - eps : float, distance maximale pour être considéré comme voisin.
    - min_samples : int, nombre minimum de points pour former un cluster.
    """
    self.eps = eps
    self.min_samples = min_samples
    self.labels_ = None  # Labels des clusters
  # Ajoutez les autres méthodes ici
```

c. Méthode fit

Implémentez la méthode fit pour exécuter l'algorithme sur les données :

```
def fit(self, X):
  """
  Exécute l'algorithme DBSCAN sur les données X.

  Paramètres:
  - X : array-like de forme (n_samples, n_features)

  Modifie:
  - self.labels_ : array de forme (n_samples,), assignation des clusters
  """
  n_samples = X.shape[0]
  self.labels_ = [None] * n_samples # Initialisation des labels
  cluster_id = 0 # Identifiant du cluster
  for point_idx in range(n_samples):
  # Votre code ici
```

d. Fonction pour Trouver les Voisins (region_query)

Implémentez une fonction pour trouver les indices des points voisins d'un point donné :

```
def region_query(self, X, point_idx):
  """
  Trouve les indices des points dans X qui sont à une distance eps du point d'indice point_idx.
```

```
    Paramètres:
    - X : array-like de forme (n_samples, n_features)
    - point_idx : int, indice du point à partir duquel chercher les voisins

    Retourne:
    - neighbors : liste des indices des points voisins
    """
    neighbors = []
    for idx in range(len(X)):
    # Votre code ici
```

e. Fonction pour Étendre le Cluster (expand_cluster)

Implémentez une fonction pour étendre le cluster en ajoutant les points densément connectés :

f. Implémentation de l'Algorithme Principal

Dans la méthode fit, implémentez l'algorithme en utilisant les fonctions précédentes :

1. Initialisez tous les points comme non visités (par exemple, avec un label None).
2. Pour chaque point non visité :
    - Marquez le point comme visité.
    - Récupérez ses voisins en utilisant region_query.
    - Si le nombre de voisins est inférieur à min_samples, marquez le point comme bruit (label = -1).
    - Sinon, attribuez un nouveau cluster et étendez-le avec expand_cluster.

g. Gestion des Labels

- Utilisez un label unique pour chaque cluster (par exemple, 0, 1, 2, ...).
- Les points de bruit doivent avoir le label -1.

Partie 3 : Application sur les Jeux de Données

3.1 Chargement des Jeux de Données

- **Iris**

- **Wine**
- **Wholesale Customers**

3.2 Prétraitement des Données

- Normalisez les données si nécessaire (utilisez StandardScaler de scikit-learn).
- Si vous utilisez StandardScaler, n'oubliez pas que vous pouvez l'utiliser car il ne fait pas partie des algorithmes de clustering.

3.3 Application de votre Implémentation

- Appliquez votre version de DBSCAN sur chaque jeu de données.
- Choisissez des valeurs appropriées pour epsilon et min_samples (vous pouvez utiliser la méthode du diagramme des distances k-plus-proches pour déterminer ε).

3.4 Visualisation des Résultats

- Pour les jeux de données avec 2 ou 3 dimensions (ou après réduction de dimensionnalité), visualisez les clusters obtenus.
- Utilisez des couleurs différentes pour représenter les clusters et identifiez les points de bruit.

Partie 4 : Comparaison avec scikit-learn

4.1 Application de scikit-learn DBSCAN

- Importez l'implémentation de DBSCAN de scikit-learn.
- Appliquez-la sur les mêmes jeux de données avec les mêmes paramètres epsilon et min_samples.

4.2 Comparaison des Résultats

- Comparez les labels obtenus par votre implémentation et celle de scikit-learn.
- Calculez des métriques de similarité, telles que le Rand Index ou l'Adjusted Rand Index.

4.3 Analyse des Performances

- Mesurez le temps d'exécution de votre algorithme et de celui de scikit-learn.
- Discutez des différences en termes de performance et de précision.

Partie 5 : Analyse et Discussion

Questions :

1. **Précision :** Votre implémentation produit-elle les mêmes clusters que scikit-learn ? Si non, pourquoi ?
2. **Performance :** Quelles sont les différences de temps d'exécution entre votre implémentation et celle de scikit-learn ? Comment pourriez-vous optimiser votre code ?
3. **Paramètres :** Comment le choix de epsilon et min_samples affecte-t-il les résultats ? Avez-vous rencontré des difficultés pour déterminer ces paramètres ?
4. **Limitations :** Quelles sont les limitations de votre implémentation ? Comment scikit-learn gère-t-il certains aspects différemment (par exemple, l'efficacité de la recherche des voisins) ?

Conclusion

- Résumez ce que vous avez appris en implémentant DBSCAN.
- Discutez de l'importance de comprendre les algorithmes sous-jacents aux outils que vous utilisez.
- Proposez des améliorations possibles pour votre implémentation.