Martin Mouly

Algorithme des K plus proches voisins

L'objectif est de déterminer la classe d'un individu à partir de ses données, en utilisant une base de données d'individus dont on connaît la classe.

J'ai tout d'abord créé une classe pour permettre de mieux gérer les data à classifier : chaque objet **Data** a pour attribut **values**, comprenant ses données (sous forme de liste), et **classe**. La méthode **distEuclide** permet de calculer la distance euclidienne entre deux objets **Data**.

La méthode **ListData** crée une liste comprenant toutes les données d'entraînement à partir d'un fichier et la méthode **FinalTestData** crée la liste des data à classifier.

La méthode **CalcDist** calcule les distances euclidiennes entre une data à classifier et toutes les data d'entraînement, puis retourne une liste de ces distances triées par ordre croissant. La méthode **Knn** permet de conserver les k plus proches voisins et la méthode **Result** permet de retourner la classe dominante des k plus proches voisins.

Pour tester le fonctionnement de l'algorithme, j'ai pris comme data d'entraînement les fichiers data.csv et preTest.csv qui me fournissent une base de données de 1606 éléments. Je split ensuite aléatoirement les données en un set d'entraînement et un set à classifier (80-20%), à l'aide de la fonction **Division**.
J'ai ensuite testé mon algorithme pour différentes valeurs de k.

```
In [62]: TestAlgo(1)
nb de données: 1606
1 voisins: 88.474 % -- temps execution: 1.025 secondes
2 voisins: 88.474 % -- temps execution: 1.017 secondes
3 voisins: 91.9 % -- temps execution: 1.027 secondes
4 voisins: 90.343 % -- temps execution: 1.018 secondes
5 voisins: 89.72 % -- temps execution: 1.016 secondes
6 voisins: 89.408 % -- temps execution: 1.01 secondes
7 voisins: 89.408 % -- temps execution: 1.017 secondes
8 voisins: 89.097 % -- temps execution: 1.012 secondes
9 voisins: 89.408 % -- temps execution: 1.009 secondes
10 voisins: 88.785 % -- temps execution: 1.011 secondes
11 voisins: 88.474 % -- temps execution: 1.016 secondes
12 voisins: 88.785 % -- temps execution: 1.03 secondes
13 voisins: 88.474 % -- temps execution: 1.023 secondes
14 voisins: 88.474 % -- temps execution: 1.023 secondes
15 voisins: 87.85 % -- temps execution: 1.011 secondes
```

Pour la suite j'ai choisi de garder k entre 3 et 7.

Afin d'améliorer la précision de mon algorithme, j'ai décidé d'utiliser une méthode de scaling des données: j'ai utilisé dans la méthode **Scaling** les min et max des 6 variables des données pour permettre une meilleure représentation et une meilleure précision.

```
In [72]: TestAlgo(3)
nb de données: 1606
3 voisins: 90.966 % -- temps execution: 1.022 secondes
4 voisins: 92.835 % -- temps execution: 1.001 secondes
5 voisins: 91.589 % -- temps execution: 1.022 secondes
6 voisins: 91.589 % -- temps execution: 1.02 secondes
7 voisins: 90.966 % -- temps execution: 1.023 secondes
```

Sans le scaling, les précisions tournaient autour de 89-90% pour 3 à 7 voisins, et avec le scaling, les précisions dépassaient souvent les 90% et pouvaient atteindre les 92%, le tout sans compromettre le temps d'exécution relativement satisfaisant d'environ 1 seconde.

Pour la classification de finalTest j'ai finalement choisi k=6 après plusieurs tests. Le temps d'exécution pour classifier les 3000 datas du fichier finalTest.csv est d'environ 11,6 secondes.