Clustering basé sur la Densité

1. Ensemble de données Data Produit QF

Nous allons durant cette séance utiliser l'ensemble de données *Data_Produit_QF.csv* qui contient des données sur l'achat d'un produit qui a été proposé à des clients. La variable *Produit* indique pour chaque client s'il a ou non acheté le produit.

Caractéristiques de l'ensemble de données :

- Instances : 600, chacune décrivant un client
- Nombre de variables : 13
- Variable de classe : *Produit*
- Valeurs manquantes : aucune

Le dictionnaire des données ci-dessous indique pour chacune des 13 variables :

- son libellé (nom),
- son type (entier, décimal, booléen, catégoriel, ordinal, date, etc.),
- sa description (sémantique),
- son domaine de valeurs (intervalle [minimum, maximum] pour les variables numériques ou liste de valeurs pour les variables modales/catégorielles).

Dictionnaire des données

Variable	*Type*	*Description*	*Domaine de valeurs*
ID	Entier	Numéro identifiant du client	[12101, 12400]
Age	Entier	Age en années	[18, 67]
Sexe	Catégoriel	Sexe	Homme, Femme
Habitat	Catégoriel	Type d'habitat	Banlieue , Centre_Ville, Petite_Ville, Rural
Revenus	Entier	Revenus annuels en dollars US	[60392, 505040]
Marie	Booléen	Statut marital	Oui, Non
Enfants	Entier	Nombres d'enfants	[0, 3]
Voiture	Booléen	Possède une voiture	Oui, Non
Compte_Epargne	Booléen	Possède un compte épargne	Oui, Non
Compte_Courant	Booléen	Possède un compte courant	Oui, Non
Emprunt	Booléen	Emprunt en cours	Oui, Non
Quotient_Familial	Entier	Rapport entre revenus et nombre d'enfants	[20280, 492400]
Produit	Booléen	Client acquéreur du produit Variable de classe	Oui, Non

Objectif

L'objectif de cette application est d'essayer d'identifier parmi chacune des deux classes, Produit = Oui et Produit = Non, des sous-groupes de clients, c-à-d un cluster de clients, ayant des caractéristiques communes (âge moyen, nombre d'enfants, etc.) qui leur sont spécifiques, c-à-d qui sont différentes de celles autres sous-groupes de la même classe.

Pour cela, plusieurs algorithmes de clustering (*K-means*, *Agnes*, etc.) seront appliqués, avec pour objectif de comparer la création de différents clusterings, chacun correspondant à un nombre spécifique de clusters (paramètre *K*) pour un algorithme particulier (ex : *K-means* pour *K* = 4).

Chargement des données

► Chargez les données du fichier *Data_Produit_QF.csv* dans un data frame `produit`.

► Supprimez la variable `ID` du data frame `produit` avec l'opérateur de sélection `[,]`.

- ► Modifiez le type de la variable *Enfants* du data frame `produit` pour le type ordinal (valeurs catégorielles ordonnée).
- ► Vérifiez le résultat de l'opération en affichant la classe et le mode de la variable `Enfants` à l'aide des fonctions `class()` et `mode()`.
- ► Affichez les caractéristiques résumées du data frame `produit` par la fonction `str()`.
- ► Affichez les caractéristiques résumées des variables du data frame `produit` par la fonction `summary()`.

2. Matrice de distance

Nous allons utiliser la librairie *cluster* qui fournit un ensemble de méthodes pour le clustering.

- ► Chargez la librairie *cluster*.

Nous allons calculer une matrice de distance pour les instances du data frame `produit` en utilisant la fonction `daisy()` du package `cluster` qui permet de traiter les données hétérogènes (numériques, catégorielles, ordinales, etc.).

- ► Calculez la matrice de distance `dmatrix` à l'aide de la fonction `daisy()`.
- ► Affichez les informations résumées de la matrice de distance `dmatrix` à l'aide de la fonction `summary()`.

3. Clustering basé sur la densité

Nous allons utiliser la librairie *dbscan* qui fournit un ensemble de méthodes pour le clustering basé sur la densité.

- ► Installez et chargez la librairie *dbscan*.

Clustering par algorithme DBScan

La fonction `dbscan()` de la libraire *dbscan* permet de réaliser un clustering basé sur la variation de la densité dans l'espace des données à partir d'un data frame (contenant des variables numériques uniquement) ou d'une matrice de distance.

- ► Exécutez le clustering par `dbscan()` à partir de la matrice de distance `dmatrix` pour une distance de voisinage (`eps`) de 0.125 et une taille minimale de voisinage (`minPts`) de 3 en stockant le résultat dans un objet `dbs` par la commande :

```
> dbs <- dbscan(dmatrix, eps=0.125, minPts = 3)
```

- ► Affichez une table de contingence affichant pour chaque cluster le nombre d'instances de chaque classe par la commande :

```
> table(dbs$cluster, poduit$Produit)
```

- ► Identifiez le nombre de clusters générés pour ces valeurs des paramètres.
- ► Affichez un histogramme d'effectifs pour chaque cluster (variable `dbs$cluster`) avec la proportion de chaque classe en couleur (variable `produit$Produit`) :

```
> qplot(as.factor(dbs$cluster), data=produit, fill=Produit)
```

4. Évaluation interne des clusters par t-SNE

Dans le cas où nous disposons d'une variable de classe dans les données permettant d'évaluer la pertinence des clusters générés, nous parlons d'*évaluation externe* des clusters.

Par opposition, si nous ne disposons pas d'une variable de classe dans les données, nous parlons d'*évaluation interne* des clusters.

L'algorithme t-SNE (t-distributed Stochastic Neighbor Embedding) est une méthode de réduction de dimension pour la visualisation de données. Cette méthode non linéaire permet de représenter un ensemble de points d'un espace à grande dimension dans un espace de deux ou trois dimensions. Il est ainsi possible de visualiser sous forme bidimensionnelle ou tridimensionnelle les résultats d'un clustering en colorant chaque point en fonction du numéro de son cluster d'affectation.

- ► Installez et chargez la librairie *tsne*.
- ► Calculer une représentation en deux dimensions, i.e. deux composantes principales, des données du data frame `produit` à partir de la matrice de distance `dmatrix` par la commande :

```
> tsne_out <- tsne(dmatrix, k=2)
```

► Convertissez le résultat en en structure de type data frame nommée `tsne_out` les coordonnées calculées par la commande :

```
> tsne_out <- data.frame(tsne_out)
```

► Afficher le nuage de points à partir des composantes calculées avec le numéro de cluster `dbs$cluster` en couleur par la commande :

```
> qplot(tsne_out[,1], tsne_out[,2], col=as.factor(dbs$cluster))
```

Plus les points appartenant à un même cluster sont regroupés dans le nuage de points, plus ce cluster obtenu est compact et bien délimité.

5. Variation de la taille du voisinage

Nous allons comparer les résultats obtenus pour différentes tailles de voisinage, c-à-d différent nombres minimaux de voisins pour former un cluster. La paramètre de distance de voisinage `eps` sera fixé à `0.125`.

► Créez une boucle `for()` qui permet pour un nombre minimal `m` de voisins variant de 3 à 8 de :

- Exécutez la fonction `dbscan()` en donnant au paramètre `minPts` la valeur `m`.
- Créez la table de contingence affichant pour chaque cluster le nombre d'instances dans chaque classe.
- Afficher l'histogramme d'effectifs pour chaque cluster avec la proportion de chaque classe en couleur.
- Afficher le nuage de points à partir des composantes calculées avec le numéro de cluster (appelé `cluster` dans le résultat généré) en couleur.

► À partir de l'histogramme d'effectifs et du nuage de points avec le numéro de cluster en couleur, identifiez le meilleur clustering obtenu, i.e. le nombre de clusters pour lequel la proportion d'instances de même classe dans chaque cluster est maximale et pour lequel les points appartenant au même cluster sont les mieux regroupés dans le nuage de points.

Les clusterings obtenus vous paraissent-ils pertinents vis à vis de leur taille, de leur proportion d'instances de même classe dans chaque cluster et de leur compacité dans le nuage de points affiché ?

6. Autres algorithmes de clustering par densité

Nous allons tester les fonctions `optics()` et `hdbscan()` de la librairie *dbscan* qui sont des algorithmes basés sur la notion de densité qui incluent des variantes dans les méthodes de calculs de voisinages et regroupements.

► Consultez la documentation des fonctions *optics()* et *hdbscan() de* la librairie *dbscan*.

► Créez une boucle `for()` qui permet pour un nombre minimal `m` de voisins variant de 3 à 8 de :

- Exécutez la fonction `optics()` en donnant au paramètre `minPts` la valeur `m`.
- Créez la table de contingence affichant pour chaque cluster le nombre d'instances dans chaque classe.
- Afficher l'histogramme d'effectifs pour chaque cluster avec la proportion de chaque classe en couleur.

► Créez une boucle `for()` qui permet pour un nombre minimal `m` de voisins variant de 3 à 8 de :

- Exécutez la fonction `hdbscan()` en donnant au paramètre `minPts` la valeur `m`.
- Créez la table de contingence affichant pour chaque cluster le nombre d'instances dans chaque classe.
- Afficher l'histogramme d'effectifs pour chaque cluster avec la proportion de chaque classe en couleur.

► Comparer tous les clusterings obtenus avec ces fonctions de la librairie *dbcan* et identifiez lequel (ou lesquels si plusieurs sont identiques) vous paraissent les plus pertinents.