Module 7
Exemples d'applications de la science des données

Sommaire

Dernière mise à jour le 25 février 2019

Introduction

La science des données est un domaine émergent dans tous les domaines. Parmi les plus grands utilisateurs de la science des données on trouves, selon Wikipédia[1] :

— Aéronautique
— Automobile
— Agriculture
— Assurance
— Banque & finance, dont "Trading financier"
— Distribution
— Économétrie, économie
— Énergie
— Industrie manufacturière
— Médias (ex : journalisme de données) & loisirs
— Météorologie
— Moteurs de recherche
— Services (industrie des services)
— Santé publique (ex : épidémiologie, toxicologie, écotoxicologie...)
— Télécommunications
— Tourisme
— Transport
— Urbanisme, smart cities, smart grids
— Publicité, e-commerce
— Environnement/Climat

1. https://fr.wikipedia.org/wiki/Science_des_données

Dans ce texte, nous allons présenter quelques exemples d'applications de la science des données pour analyser des données dans certains domaines sélectionnés. L'objectif est de montrer comment les techniques de la science des données peuvent être appliquées à différents types de données dans différents domaines. Nous avons sélectionné trois domaines à savoir les réseaux sociaux, l'environnement, et l'intelligence d'affaires pour présenter nos exemples.

7.1 Exemple d'application dans le domaine des réseaux sociaux

Les réseaux sociaux tels que Twitter, Facebook, Youtube, etc. représentent de grands utilisateurs de la science des données, et les compagnies qui détiennent ces réseaux sociaux contribuent grandement au développement des techniques et algorithmes de la science des données. Le grand volume de données qui transitent par ces réseaux sociaux crée un grand besoin pour analyser et comprendre ces données. Les données provenant des réseaux sociaux représentent un grand défis d'analyse vu leur variété (textes, images, vidéos) et leur volume. En plus de leur utilisation comme un moyen de communication et d'échange, les réseaux sociaux sont utilisés également par les compagnies de marketing, et les publicités en ligne. Cela rend important de comprendre comment les réseaux sociaux sont structurés et comment les utilisateurs sont organisés (groupes, communautés, etc.) pour pouvoir cibler des publicités par exemple ou recommander des produits.
Nous allons présenter, dans ce qui suit, un exemple de données provenant des réseaux sociaux, et comment ces données seront analysées pour extraire des informations pertinentes sur la structure du réseaux et les interactions entre les utilisateurs.

7.1.1 Données des réseaux sociaux

Il existe beaucoup de données relatives aux réseaux sociaux sur le web, notamment sur le site https://snap.stanford.edu/data/. Nous pouvons citer par exemple :

— Données de Facebook https://snap.stanford.edu/data/egonets-Facebook.html

— Données de Youtube https://snap.stanford.edu/data/com-Youtube.html

— Données d'Amazon https://snap.stanford.edu/data/com-Amazon.html

— Données de LiveJournal https://snap.stanford.edu/data/com-LiveJournal.html

— Données de Google+ https://snap.stanford.edu/data/egonets-Gplus.html

— Donées de Twitter https://snap.stanford.edu/data/egonets-Twitter.html

Dans ce cours, nous allons choisir un ensembles de données, Facebook, pour explorer les données. Les réseaux sociaux sont généralement représentés sous forme de graphes, dirigés ou non dirigés, où les noeuds représentent les utilisateurs, et les arrêtes (arcs) représentent les interactions entre les utilisateurs. Comme les données de Facebook ne permettent pas de savoir qui suit qui, c'est-à-dire, quel utilisateur suit quel autre utilisateur, donc ces données peuvent être représentées sous forme de graphe non dirigé. Les techniques utilisées dans ce contexte peuvent également être utilisées dans le contexte des graphes dirigés (comme dans le cas de Twitter).

7.1.2 Données de Facebook

Nous allons présenter les données de Facebook que vous pouvez obtenir à partir du lien suivant https://snap.stanford.edu/data/egonets-Facebook.html. Ces données contiennent des informations sur des utilisateurs, leur caractéristiques, leurs interactions, et leurs regroupements.

7.1.3 Préparation des données

Tout d'abord, nous allons présenter comment ces données sont importées dans R. Une fois les données importées, nous allons voir comment les représenter sous forme de graphe non dirigé. Les données de Facebook sont organisées en plusieurs fichiers. Nous allons utiliser un seul fichier avec l'extension .edges qui stocke toutes les arrêtes d'un noeud

donné. Ces arrêtes, comme nous l'avons mentionné, représentent les interactions de ce noeud avec tous les autres noeuds dans le fichier. Ces fichiers stockent les données sous forme de graphe. Nous allons donc faire appel à la fonction `read.graph` de la librairie `igraph` de R pour ces exemples.
Dans l'exemple suivant, nous allons charger ces données dans R en choisissant un noeud au hasard, soit le noeud **0**, donc le fichier **0.edges**.

```
# Lecture du fichier 0.edges
MyGraphData <- read.graph(file = "./facebook/0.edges", directed =↩
    FALSE)
```

La fonction `read.graph` prend en paramètre le nom du fichier, dans ce cas "0.edges", et un paramètre, `directed`, qui indique si le graphe est dirigé ou non. Dans le cas de Facebook le graphe n'est pas dirigé. Une fois que le graphe est chargé, nous pouvons faire beaucoup d'opérations, à savoir compter le nombre de noeuds et le nombre d'arrêtes, vérifier la structure des noeuds, calculer la mesure de centralité d'un noeud dans le graphe, etc.
Nous pouvons vérifier le nombre de noeuds (qui représentent des usagers de Facebook) et d'arrêtes (qui représentent des liens d'amitié) dans le graphe comme suit :

```
# Nombre de noeuds dans le graphe
Nb_noeuds <- vcount(MyGraphData)
> Nb_noeuds
[1] 348

# Pour enlever les doublons et les boucles dans le graphe
MyGraphData <- simplify(MyGraphData, remove.multiple = TRUE, ↩
   remove.loops = TRUE)

# Nombre d'arrêtes dans le graphe
Nb_arretes <- ecount(MyGraphData)
> Nb_arretes
[1] 2519
```

Nous pouvons également visualiser la structure hiérarchique des noeuds qui forment des communautés dans le graphe comme suit :

```
# Structure hiérarchique des noeuds
cluster <- edge.betweenness.community(MyGraphData)
> cluster          # les noeuds dans chaque communauté
$`1`
  [1] 1

  $`2`
   [1]   2  25  49  54  58  74  81  93  95 102 127 131 181 188 ↩
       192 195 197
  [18] 205 243 250 255 267 300 301 303 331 347

  $`3`
   [1]   3  15  18  20  21  29  33  42  45  94 112 113 116 117 ↩
       138 139 141
  [18] 145 150 152 163 168 175 215 217 221 227 244 263 280 290 ↩
      294 306 311
  ...

# Visualisation
plot_dendrogram(cluster) # voir figure en bas
```

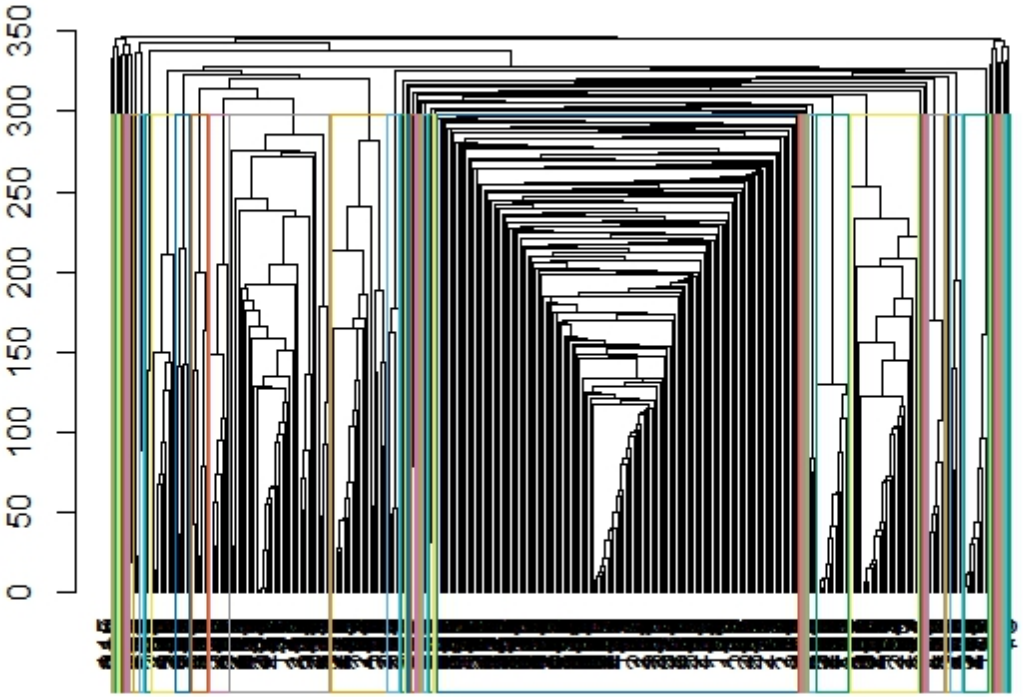


Figure 7.1: Structure hiérarchique des noeuds qui forment des communautés. Nous pouvons constater que les communautés sont beaucoup plus denses au milieu du graphe, moins denses à gauche, et beaucoup moins denses à droite du graphe.

Nous pouvons visualiser aussi le graphe au complet pour voir comment les noeuds sont

interconnectés comme suit :

```
# Le seed est utiliser pour reproduire les mêmes résultats
set.seed("20180507")
# Ajouter la communauté indiquant les couleurs d'arrière plan
plot(MyGraphData, vertex.color = cluster$membership, vertex.size = log(degree(MyGraphData) + 1),
     mark.groups = by(seq_along(cluster$membership), cluster$membership, invisible))

# Ajouter les annotations
title("Visualisation du graphe complet")
```

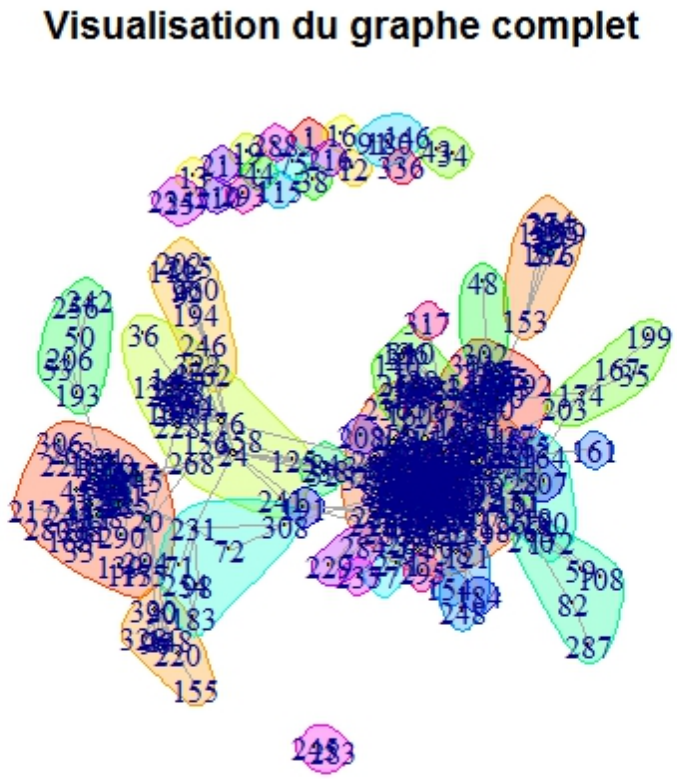


Figure 7.2: Visualisation du graphe au complet. Dans cette représentation, il est claire que dans la partie droite du graphe, les communautés sont beaucoup plus denses, et donc les interactions entre les utilisateurs sont importantes, contrairement aux autres parties du graphe.

Centralité d'un noeud dans un graphe

La mesure de centralité consiste à mesurer l'importance des noeuds dans un graphe. L'une des applications importantes de cette mesure incluent l'identification des utilisateurs les plus influents dans les réseaux sociaux. Notez qu'il existe plusieurs mesures de centralité telle que la centralité intermédiaire, la centralité de proximité, etc. Nous allons présenter un seul exemple dans ce cours, c'est la centralité intermédiaire qui est la plus utilisée.
La centralité intermédiaire (en anglais, *betweenness centrality*) d'un noeud A dans un graphe est égale au nombre de fois que ce noeud est sur le chemin le plus court entre deux autres noeuds quelconques B et C du graphe[2]. La formule de calcul de cette mesure est comme suit :

$$Central(A) = \sum_{A \neq B \neq C} \frac{\sigma_{BC}(A)}{\sigma_{BC}} \tag{7.1}$$

Tel que σ_{BC} représente le nombre total des chemins les plus courts entre les deux noeuds B et C, et $\sigma_{BC}(A)$ représente le nombre total des chemins les plus courts entre les deux noeuds B et C qui passent par le noeud A. Plus la valeur de centralité s'approche de 1, plus le noeud devient central est donc, le noeud aura plus d'importance dans le graphe. Avec les données de Facebook, nous pouvons vérifier calculer la centralité intermédiaire moyenne dans tout le graphe comme suit :

```
# Centralité intermédiaire de chaque noeud
C_i <- betweenness(igraphDat)
C_i_norm <- (C_i - min(C_i) /(max(C_i) - min(C_i)) # ↩
   Normalisation entre 0 et 1 comme nous pouvons le voir dans la ↩
   figure suivante:

plot(C_i_norm, xlab="Noeuds", ylab="Centralité intermédiaire", ↩
   col ="red")
```

2. https://fr.wikipedia.org/wiki/Centralit%C3%A9_interm%C3%A9diaire

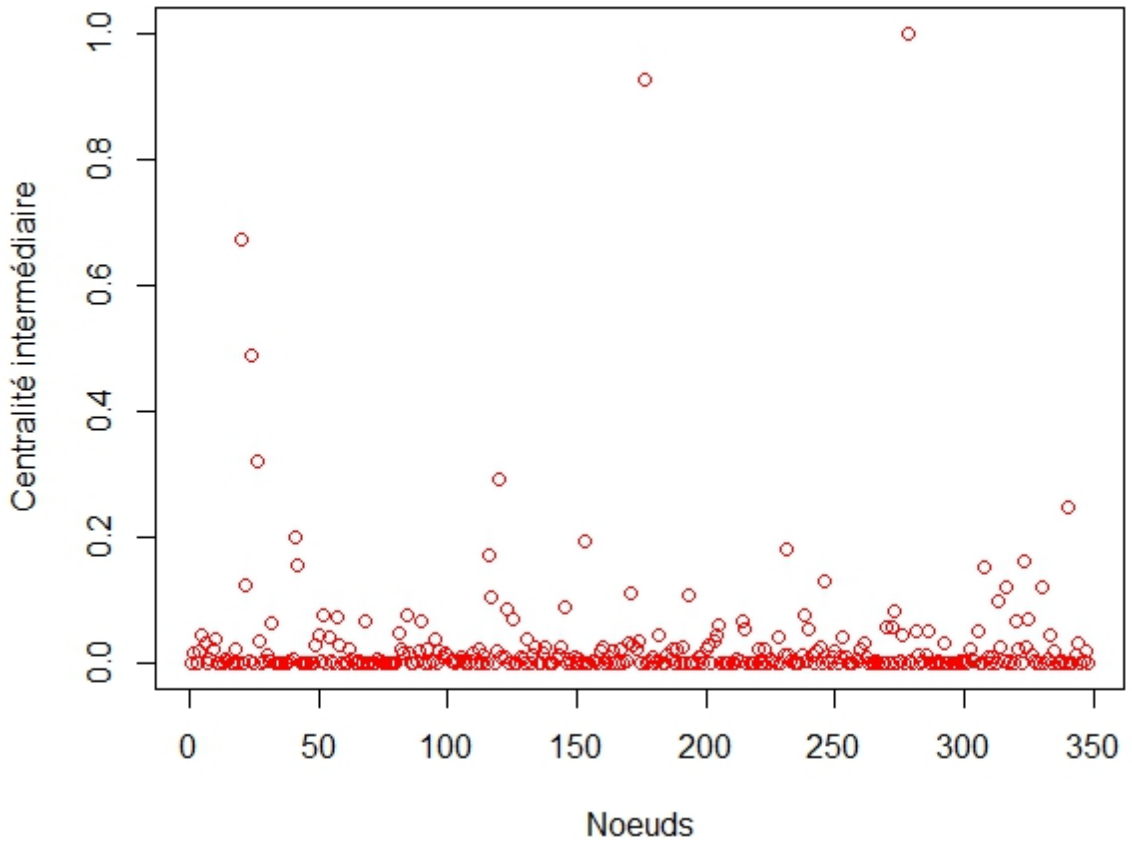


Figure 7.3: Centralité intermédiaire pour chaque noeud dans le graphe. Nous pouvons constater que la majorité des noeuds ont une centralité très petite, et seulement quelques noeuds qui possèdent des valeurs de centralité plus élevées, et donc sont considérés comme des noeuds importants et donc des utilisateurs influents de Facebook.

Nous pouvons aussi calculer la centralité moyenne dans le graphe comme suit :

```
# Centralité intermédiaire moyenne
C_i_moyenne <- mean(C_i_norm)

> C_i_moyenne
[1] 0.02833488
```

Comme nous pouvons le voir, la centralité intermédiaire moyenne dans le graphe est égale à 0.028. Cela est logique car rares sont les noeuds centraux et, généralement, le nombre des utilisateurs influents dans un réseaux social est très petit. Pour choisir les noeuds centraux, nous pouvons sélectionner ceux ayant une centralité intermédiaire supérieure à 0.5 par exemple. Dans notre cas, nous avons :

```
> length(C_i_norm[C_i_norm > 0.5])
[1] 3
```

Donc, nous avons uniquement 3 noeuds centraux selon notre critère de sélection. Nous pouvons avoir plus de noeuds si nous modifions le critère de sélection en mettant des valeurs plus basses 0.2, 0.3, 0.4 etc.

7.2 Exemple d'application dans le domaine de l'environnement

Beaucoup de défis auxquels nous sommes confrontés aujourd'hui proviennent de la compréhension et de l'interaction avec le monde naturel comme la compréhension des changements climatiques, la propagation des maladies et des espèces envahissantes, et ainsi de suite. Avec le progrès technologique, et l'apparition des satellites, capteurs et internet des objets, nous disposons présentement de quantités énormes de données sur le monde naturel. Ces données nécessitent d'être explorées afin de les comprendre et d'en extraire des informations pertinentes qui peuvent nous aider à faire face aux différents défis auxquels

nous sommes confrontés.
Dans cette section, nous allons analyser des données météorologiques de Montréal. Ces données sont disponibles pour téléchargement à partir du site web du gouvernement du Canada au http://climate.weather.gc.ca/. Nous avons choisi la station de l'aéroport de Montréal parce qu'elle contient des données sur la température depuis plusieurs années. Nous avons pris les données de 2010 au 2018.
Nous commençons d'abord par charger le fichier :

```
# Lire le fichier csv
df <- read.csv(file = "Montreal_meteo.csv", header = T, sep = ","↩
    )
```

L'objet dataframe contient donc 3048 observations et 27 attributs (*features*). Ces informations peuvent être obtenues comme suit :

```
> dim(df)
[1] 3048   27
```

La liste des attributs utilisés sont :

```
> colnames(df)
 [1] "Date.Time"                "Year"
 [3] "Month"                    "Day"
 [5] "Data.Quality"             "Max.Temp"
 [7] "Max.Temp.Flag"            "Min.Temp"
 [9] "Min.Temp.Flag"            "Mean.Temp"
[11] "Mean.Temp.Flag"           "Heat.Deg.Days"
[13] "Heat.Deg.Days.Flag"       "Cool.Deg.Days"
[15] "Cool.Deg.Days.Flag"       "Total.Rain..mm."
[17] "Total.Rain.Flag"          "Total.Snow..cm."
[19] "Total.Snow.Flag"          "Total.Precip..mm."
[21] "Total.Precip.Flag"        "Snow.on.Grnd..cm."
[23] "Snow.on.Grnd.Flag"        "Dir.of.Max.Gust..10s.deg."
[25] "Dir.of.Max.Gust.Flag"     "Spd.of.Max.Gust"
```

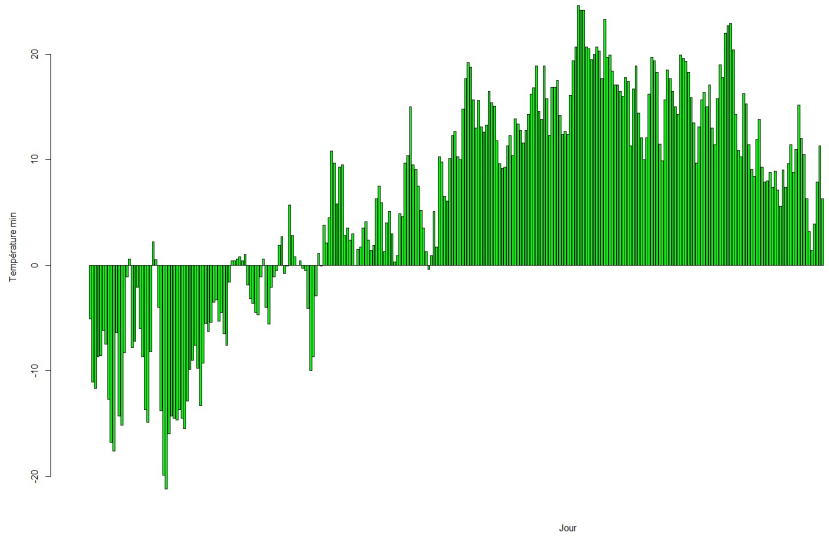
```
[27] "Spd.of.Max.Gust.Flag"
```

Nous pouvons également vérifier les périodes de collecte des données comme suit :

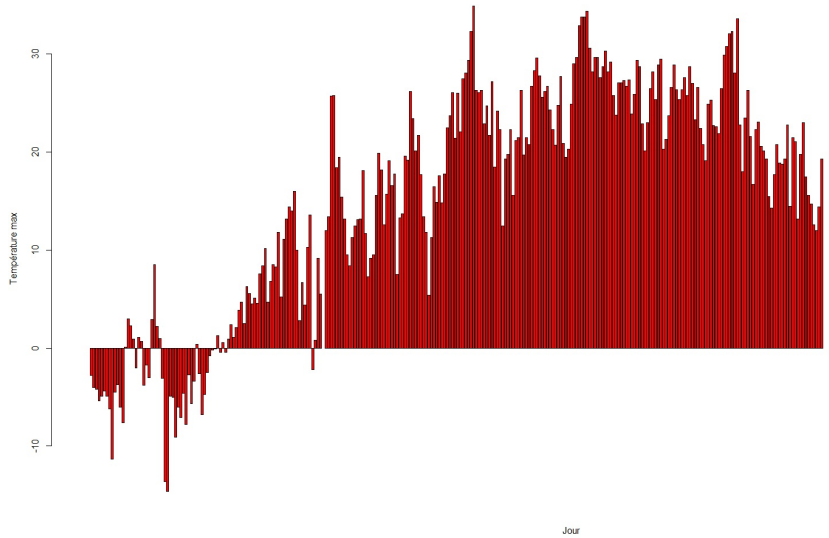
```
# Années
> unique(df$Year)
[1] 2018 2017 2016 2015 2014 2013 2012 2011 2010

# Mois
> unique(df$Month)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

Donc, les données sont collectées sur une période allant de 2010 au 2018 pour les 12 mois de l'année.
Dans cette analyse, nous sommes intéressés à l'étude de la température et ses variations durant les années. Nous pouvons visualiser les températures minimales et maximales pour l'année 2010 comme suite :

```
# Données 2010
df_2010 <-  subset(df, df$Year == "2010")

barplot(df_2010$Min.Temp, col = "green", xlab = "Jour", ylab = "↩
    Température min") # Voir figure 7.4(a)

barplot(df_2010$Max.Temp, col = "red", xlab = "Jour", ylab = "↩
    Température max") # Voir figure 7.4(b)
```



(a) Températures minimales 2010



(b) Températures maximales 2010

Figure 7.4: Températures minimales et maximales par jours pour l'année 2010.

Nous pouvons également visualiser les températures moyennes pour toutes les années comme suit :

```
barplot(df$Mean.Temp, col = "blue", xlab = "Jour", ylab = "↩
    Température moyenne") # Voir figure 7.5
```

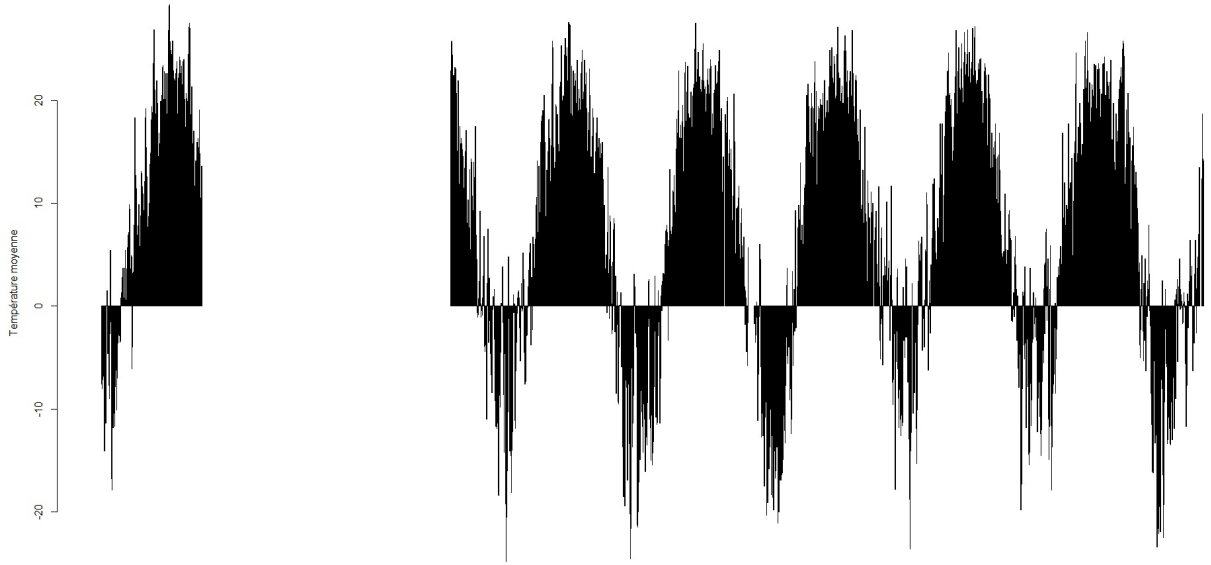


Figure 7.5: Températures moyennes pour toutes les années.

Comme nous pouvons le voir dans la figure 7.5, il existe un trou dans les données. C'est une période d'octobre 2010 jusqu'au mois d'août 2012. Cela démontre aussi l'intérêt de la visualisation pour explorer les données.
Il existe beaucoup d'autres attributs qui mériteraient d'être explorés comme la vitesse du vent et sa corrélation avec d'autres variables. Nous pouvons visualiser la vitesse du vent pour toutes les années comme suit :


```r
barplot(as.numeric(df$Spd.of.Max.Gust), xlab = "Jour", ylab = "↩
    Vitesse du vent") # Voir figure 7.6
```

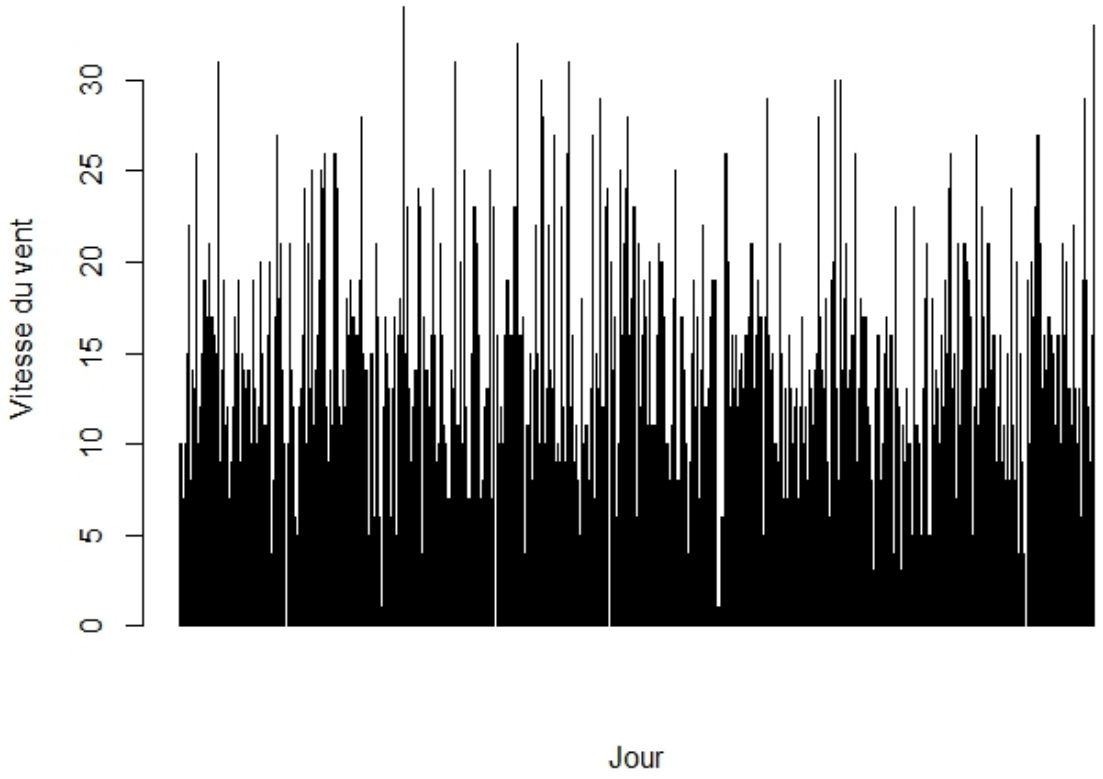


Figure 7.6: Vitesse du vent pour toutes les années.

7.2.1 Corrélation

Il est important d'explorer également les corrélations entres les différentes variables sur le même ensemble de données météorologiques. La corrélation peut révéler des informations pertinentes sur les variables et nous aider à bien comprendre les relations entres les différentes variables. Nous allons utiliser la corrélation de Pearson[3]. Pour ce faire, nous allons utiliser la fonction `cor` de la librairie `ggpubr`. Nous allons essayer de chercher des corrélations entre plusieurs variables prises deux à deux.

```
# Corrélation de Pearson
library("ggpubr")

# Corrélation entre vitesse du vent et température maximale
corr_spd_max_temp <- cor(as.numeric(df$Spd.of.Max.Gust), as.↩
    numeric(df$Max.Temp),  method = "pearson", use = "complete.obs↩
    ")

> corr_spd_max_temp
[1] -0.03870076

# Corrélation entre vitesse du vent et température minimale
corr_spd_min_temp <- cor(as.numeric(df$Spd.of.Max.Gust), as.↩
    numeric(df$Min.Temp),  method = "pearson", use = "complete.obs↩
    ")

> corr_spd_min_temp
[1] -0.07110144

# Corrélation entre vitesse du vent et température moyenne
corr_spd_mean_temp <- cor(as.numeric(df$Spd.of.Max.Gust), as.↩
    numeric(df$Mean.Temp),  method = "pearson", use = "complete.↩
    obs")
```

3. https://fr.wikipedia.org/wiki/Corr%C3%A9lation_(statistiques)

```
> corr_spd_mean_temp
[1] -0.05493099
```

Comme nous pouvons le constater, les corrélations sont très faibles, voire nulles, entre ces attributs. Nous ne pouvons pas bâtir de conclusions à partir de ces corrélations, mais ce que nous savons est que les températures sont corrélées avec la vitesse du vent en hiver par exemple. Nous pouvons vérifier cette réalité en effectuant une analyse saisonnière en choisissant par exemple les données de l'hiver, en particulier les mois de décembre, janvier et février comme suit :

```
df_hiver <- subset(df, df$Month %in% c("1", "2", "12"))

# Corrélation entre vitesse du vent et température maximale
corr_spd_max_temp_hiver <- cor(as.numeric(df_hiver$Spd.of.Max.↩
    Gust), as.numeric(df_hiver$Max.Temp),  method = "pearson", use↩
     = "complete.obs")

> corr_spd_max_temp_hiver
[1] 0.2289772

# Corrélation entre vitesse du vent et température minimale
corr_spd_min_temp_hiver <- cor(as.numeric(df_hiver$Spd.of.Max.↩
    Gust), as.numeric(df_hiver$Min.Temp),  method = "pearson", use↩
     = "complete.obs")

> corr_spd_min_temp_hiver
[1] 0.1104909

# Corrélation entre vitesse du vent et température moyenne
corr_spd_mean_temp_hiver <- cor(as.numeric(df_hiver$Spd.of.Max.↩
    Gust), as.numeric(df_hiver$Mean.Temp),  method = "pearson", ↩
    use = "complete.obs")

> corr_spd_mean_temp_hiver
[1] 0.1766229
```

Comme nous pouvons le voir, les valeurs de corrélations augmentent pour la saison d'hiver. Une corrélation de 0.228 est obtenue entre la vitesse du vent et la température maximale. Nous pouvons chercher d'autres informations intéressantes dans les données saisonnières comme :

```
# Le minimum des températures minimales
> min(df_hiver$Min.Temp, na.rm = T)
[1] -27.6

# Le maximum des températures minimales
> max(df_hiver$Min.Temp, na.rm = T)
[1] 6.2

# Le minimum des températures maximales
> min(df_hiver$Max.Temp, na.rm = T)
[1] -22.3

# Le maximum des températures maximales
> max(df_hiver$Max.Temp, na.rm = T)
[1] 17.2

# La moyenne des températures
> mean(df_hiver$Mean.Temp, na.rm = T)
[1] -6.600172

# L'écart type des températures minimales
> sd(df_hiver$Min.Temp, na.rm = T)
[1] 7.649386

# L'écart type des températures maximales
> sd(df_hiver$Max.Temp, na.rm = T)
[1] 6.770659
```

Nous pouvons faire la même chose avec les données de l'été en choisissant les trois mois d'été soit le mois de juin, juillet et août comme suit ;

```
df_ete <- subset(df, df$Month %in% c("6", "7","8"))

# Corrélation entre vitesse du vent et température maximale
corr_spd_max_temp_ete <- cor(as.numeric(df_ete$Spd.of.Max.Gust), ↩
    as.numeric(df_ete$Max.Temp),  method = "pearson", use = "↩
    complete.obs")

> corr_spd_max_temp_ete
[1] 0.1158579

# Corrélation entre vitesse du vent et température minimale
corr_spd_min_temp_ete <- cor(as.numeric(df_ete$Spd.of.Max.Gust), ↩
    as.numeric(df_ete$Min.Temp),  method = "pearson", use = "↩
    complete.obs")

> corr_spd_min_temp_ete
[1] 0.1223089

# Corrélation entre vitesse du vent et température moyenne
corr_spd_mean_temp_ete <- cor(as.numeric(df_ete$Spd.of.Max.Gust),↩
     as.numeric(df_ete$Mean.Temp),  method = "pearson", use = "↩
    complete.obs")

> corr_spd_mean_temp_ete
[1] 0.1273108
```

Les corrélations sont moins importantes pour la saison d'été où une corrélation de 0.115 est obtenue entre la vitesse du vent et la température maximale. Nous pouvons également chercher des informations intéressantes sur les données d'été comme suit :

```
# Le minimum des températures minimales
> min(df_ete$Min.Temp, na.rm = T)
```

```
[1] 3.8

# Le maximum des températures minimales
> max(df_ete$Min.Temp, na.rm = T)
[1] 24.6

# Le minimum des températures maximales
> min(df_ete$Max.Temp, na.rm = T)
[1] 12.5

# Le maximum des températures maximales
> max(df_ete$Max.Temp, na.rm = T)
[1] 34.4

# La moyenne des températures
> mean(df_ete$Mean.Temp, na.rm = T)
[1] 20.47325

# L'écart type des températures minimales
> sd(df_ete$Min.Temp, na.rm = T)
[1] 3.410024

# L'écart type des températures maximales
> sd(df_ete$Max.Temp, na.rm = T)
[1] 3.738342
```

7.3 Exemple d'application dans le domaine de l'intelligence d'affaires

L'intelligence d'affaires est l'un des domaines où la science des données est largement utilisée. Dans cette section, nous allons présenter un exemple d'analyse des données transactionnelles pour montrer quelques techniques utilisées dans l'analyse de ce type de

données. Les données transactionnelles sont maintenant disponibles un peu partout dans les magasins en ligne, les détaillants et commerçants, etc.
L'analyse des données transactionnelles se fait avec des algorithmes de forage des patrons fréquents et des règles d'association [4]. L'extraction des items fréquents repose sur deux paramètres : le support et la confiance. Le support est défini comme étant la proportion des transactions contenant l'item en question. Le support minimal est un seuil utilisé pour déterminer si un item est fréquent ou pas. Si l'item apparaît un nombre de fois plus élevé que le support minimal, il sera considéré comme fréquent. La valeur du support minimal est généralement très petite comme 0.001, 0.005, etc.
La mesure de confiance est utilisée pour vérifier la force d'une règle d'association de type $X \rightarrow Y$, c'est à dire, la proportion des transactions contenant à la fois les items X et Y. En langage probabiliste, la mesure de confiance pourrait être interprétée comme la probabilité conditionnelle $P(Y|X)$. Plus la valeur de confiance est élevée, plus la règle est forte. Les valeurs de confiance sont généralement entre 0 et 1.
Il existe également d'autres mesure comme le lift, qui représente en quelque sorte un degré de corrélation entre les items dans la règle. Soit T le Nombre total des transactions dans l'ensemble de données, ces mesures sont calculées comme suit :

$$Support(X) = \frac{frequence(X)}{T} \tag{7.2}$$

$$Confiance(X \rightarrow Y) = \frac{Support(X \cap Y)}{Support(X)} \tag{7.3}$$

$$Lift(X \rightarrow Y) = \frac{Support(X \cap Y)}{Support(X) \times Support(Y)} \tag{7.4}$$

Nous avons choisi l'ensemble de donnée `Groceries` de la librairie `arules`. Cet ensemble de données contient des informations sur l'achat des différents items dans un magasin. Donc, pour pouvoir utiliser ces données, nous devons d'abord installer la librairie `arules`

4. http ://www.charuaggarwal.net/freqbook.pdf

avec la commande `install.packages('arules')`. Nous avons besoin également de la librairie `arulesViz` pour la visualisation des résultats. Nous devons l'installer également avec la même commande `install.packages('arulesViz')`. Une fois les deux librairies installées, nous pouvons charger les données comme suit :

```
# Charger les données Groceries
data(Groceries)
```

Nous pouvons visualiser les items les plus fréquents dans les données comme suit (voir la figure 7.7) :

```
# Visualiser les items les plus fréquents dans les données
itemFrequencyPlot(Groceries, topN = 30, type = "absolute")  # ↩
    voir figure en bas
```

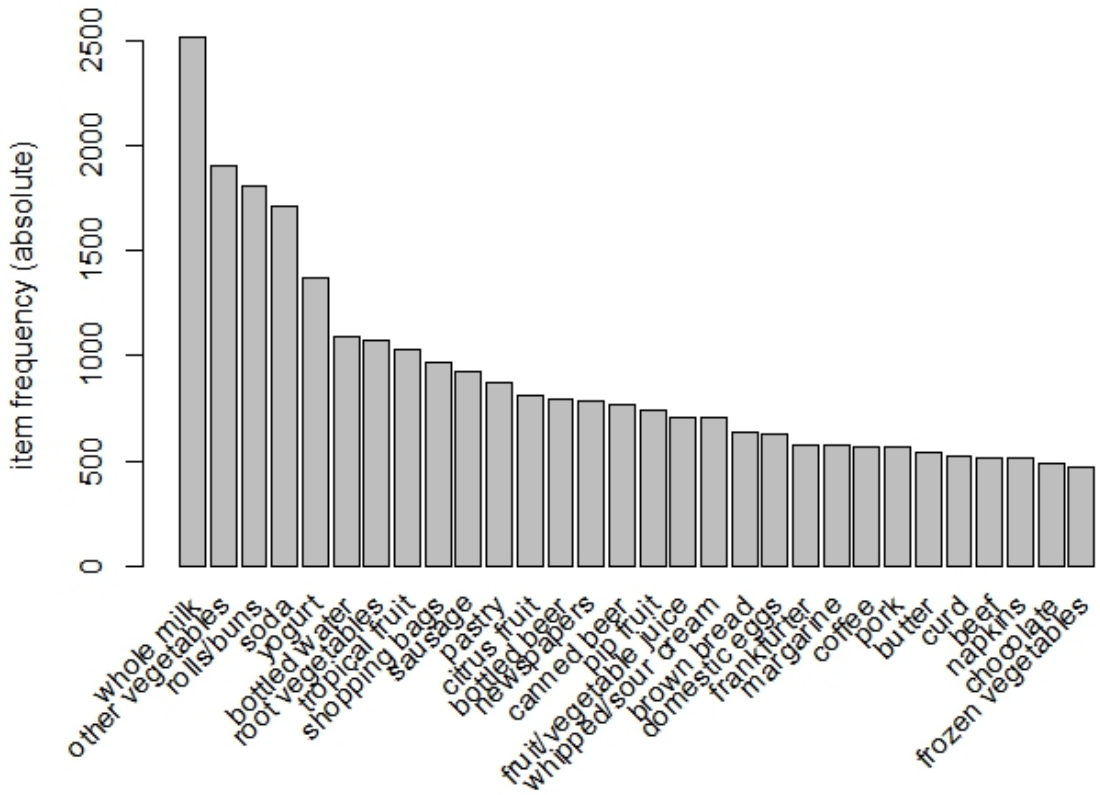


Figure 7.7: Les items les plus fréquents dans les données.

Donc, pour extraire les règles d'association, nous appliquerons l'algorithme Apriori [5]. C'est l'algorithme le plus connu et le plus utilisé pour extraire des règles d'association à partir

5. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases [archive]. Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.

des données transactionnelles. Comme nous l'avons mentionné, nous aurons besoin de définir deux paramètres : le support minimal et la confiance minimale. Nous mettons le support minimal à 0.001, et la confiance minimale à 0.8. Choisir des valeurs plus élevées permet de filtrer les règles à faible confiance et donc moins fortes.

```
# Algorithme Apriori
regles <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))

# Afficher les top 10 règles
> inspect(regles[1:10])
     lhs                           rhs                    support
        confidence      lift count
[1]  {liquor, red/blush wine} => {bottled beer}     0.001931876
   0.9047619 11.235269    19
[2]  {curd, cereals} => {whole milk}       0.001016777  0.9090909
     3.557863    10
[3]  {yogurt, cereals} => {whole milk}       0.001728521
   0.8095238  3.168192    17
[4]  {butter, jam} => {whole milk}       0.001016777  0.8333333
   3.261374    10
[5]  {soups, bottled beer} => {whole milk}       0.001118454
   0.9166667  3.587512    11
[6]  {napkins, house keeping products} => {whole milk}
   0.001321810  0.8125000  3.179840    13
[7]  {whipped/sour cream, house keeping products} => {whole milk}
          0.001220132  0.9230769  3.612599    12
[8]  {pastry, sweet spreads} => {whole milk}       0.001016777
   0.9090909  3.557863    10
[9]  {turkey, curd} => {other vegetables} 0.001220132  0.8000000
    4.134524    12
[10] {rice, sugar} => {whole milk}       0.001220132  1.0000000
   3.913649    12
```

Les règles d'association s'interprètent de la façon suivante : pour la règle {liquor, red/blush

wine} => {bottled beer}, les personnes qui achètent l'item `liquor` et `red/blush wine` conjointement, ont plus de 90% de chance d'acheter l'item `bottled beer`. Cette règle a une confiance de 0.9, donc c'est une règle très forte.
Nous pouvons également afficher un résumé de toutes les mesures comme suit :

```
# Résumé des règles
> summary(regles)
set of 410 rules

rule length distribution (lhs + rhs):sizes
  3   4   5   6
 29 229 140  12

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   4.000   4.000   4.329   5.000   6.000

summary of quality measures:
    support           confidence          lift            count
 Min.   :0.001017   Min.   :0.8000   Min.   : 3.131   Min.   ↩
    :10.00
 1st Qu.:0.001017   1st Qu.:0.8333   1st Qu.: 3.312   1st Qu↩
    .:10.00
 Median :0.001220   Median :0.8462   Median : 3.588   Median ↩
    :12.00
 Mean   :0.001247   Mean   :0.8663   Mean   : 3.951   Mean   ↩
    :12.27
 3rd Qu.:0.001322   3rd Qu.:0.9091   3rd Qu.: 4.341   3rd Qu↩
    .:13.00
 Max.   :0.003152   Max.   :1.0000   Max.   :11.235   Max.   ↩
    :31.00

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.8
```

Comme nous pouvons le voir dans le résumé, nous avons 29 règles de taille 3 (nombre d'items dans la règle, par exemple `liquor`, `red/blush wine` et `bottled beer`), 229 règles de longueur 4, 140 règles de longueur 5 et 12 règles de longueur 6. Notez que plus la longueur augmente, moins fréquentes seront les règles.
Nous pouvons visualiser toutes les règles de la façon suivante :

```
plot(regles, method="graph", engine='interactive',shading=NA) # ↩
    voir figure en bas
```

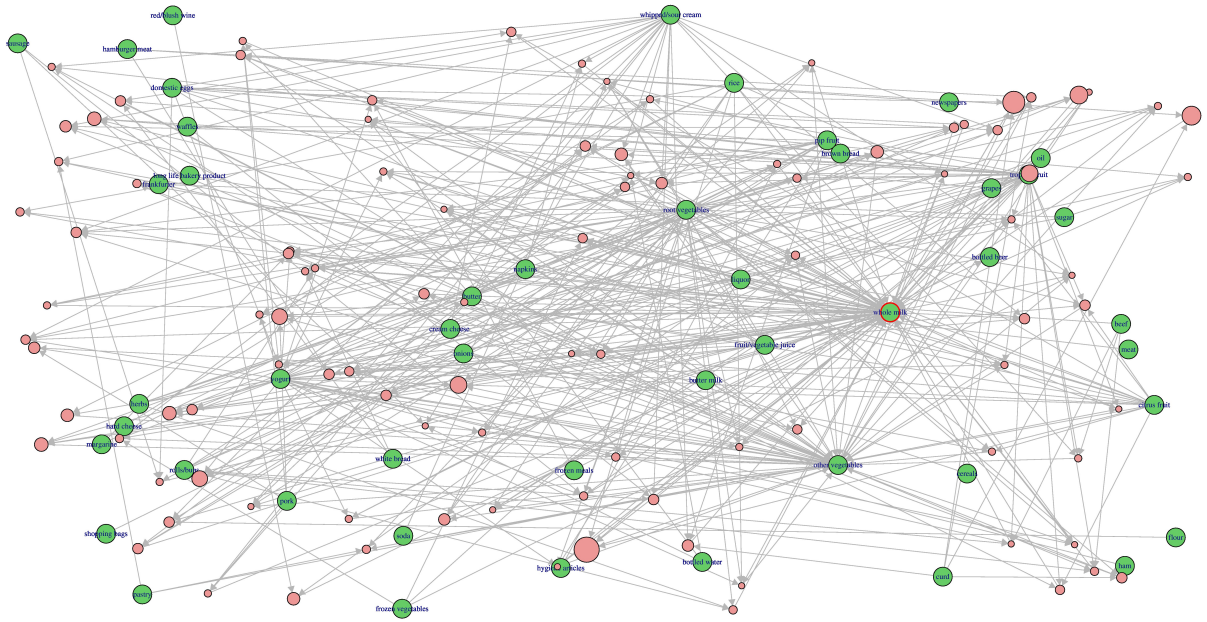


Figure 7.8: Visualisation des règles par un graphe.

7.3.1 Règles ciblées

Nous pouvons cibler des règles qui contiennent des items particuliers dans la partie de gauche (lhs) ou la partie de droite (rhs). Par exemple, nous pouvons chercher uniquement des règles qui contiennent l'item `whole milk`. Pour ce faire, nous allons modifier la fonction `apriori` pour ajouter ces informations comme suit :

```
# Règles ciblées
regles_ciblees <-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.8),
                appearance = list(default="lhs",rhs="whole milk"),
                control = list(verbose=F))

# Trier les règles dans un ordre décroissant par leur confiance
regles_ciblees <-sort(regles_ciblees, decreasing = TRUE, by = "confidence")

> inspect(regles_ciblees[1:10])
     lhs                                                rhs
                  support     confidence lift
[1]  {rice,sugar}                                    => {whole milk} 0.001220132 1          3.913649
[2]  {canned fish,hygiene articles}                  => {whole milk} 0.001118454 1          3.913649
[3]  {root vegetables,butter,rice}                   => {whole milk} 0.001016777 1          3.913649
[4]  {root vegetables,whipped/sour cream,flour}      => {whole milk} 0.001728521 1          3.913649
[5]  {butter,soft cheese,domestic eggs}              => {whole milk} 0.001016777 1          3.913649
[6]  {pip fruit,butter,hygiene articles}             => {whole milk} 0.001016777 1          3.913649
[7]  {root vegetables,whipped/sour cream,hygiene articles} => {whole milk} 0.001016777 1          3.913649
[8]  {pip fruit,root vegetables,hygiene articles}    => {
```

```
   whole milk} 0.001016777 1           3.913649
[9]  {cream cheese ,domestic eggs,sugar}                    => {
   whole milk} 0.001118454 1           3.913649
[10] {curd,domestic eggs,sugar}                             => {
   whole milk} 0.001016777 1           3.913649
     count
[1]  12
[2]  11
[3]  10
[4]  17
[5]  10
[6]  10
[7]  10
[8]  10
[9]  11
[10] 10
```

Nous pouvons également visualiser ces règles ciblées comme suit :

```
plot(regles_ciblees, method="graph", engine='interactive',shading
   =NA) # voir figure en bas
```

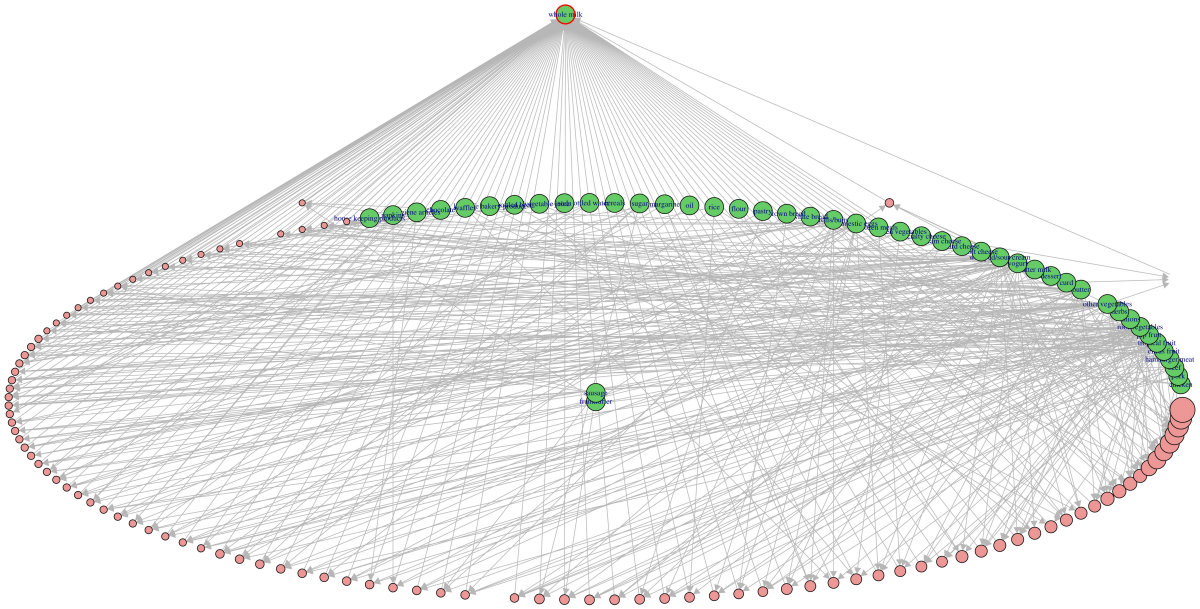


Figure 7.9: Visualisation des règles ciblées par un graphe. Ce graphe sous forme de cône, le sommet représente l'item cible, dans ce cas l'item **whole milk**.

7.3.2 Clustering des règles d'association

Un aspect important dans le forage des règles est que plusieurs règles sont similaires et donc partagent des items. Nous pouvons catégoriser ces règles par l'application d'algorithme de clustering. Pour ce faire, nous avons besoin de définir une mesure de dissimilarité, dans notre cas nous choisissons la mesure du Jaccard https://en.wikipedia.org/wiki/Jaccard_index. Nous procéderons comme suit :

```
# Définir une mesure de dissimilarité
d <- dissimilarity(regles, method = "Jaccard")

library(stats)
```

```r
cl <- kmeans(d, 5)
plot(d, col = cl$cluster)
points(cl$centers, col = 1:5, pch = 8, cex = 2) # voir figure en bas
```

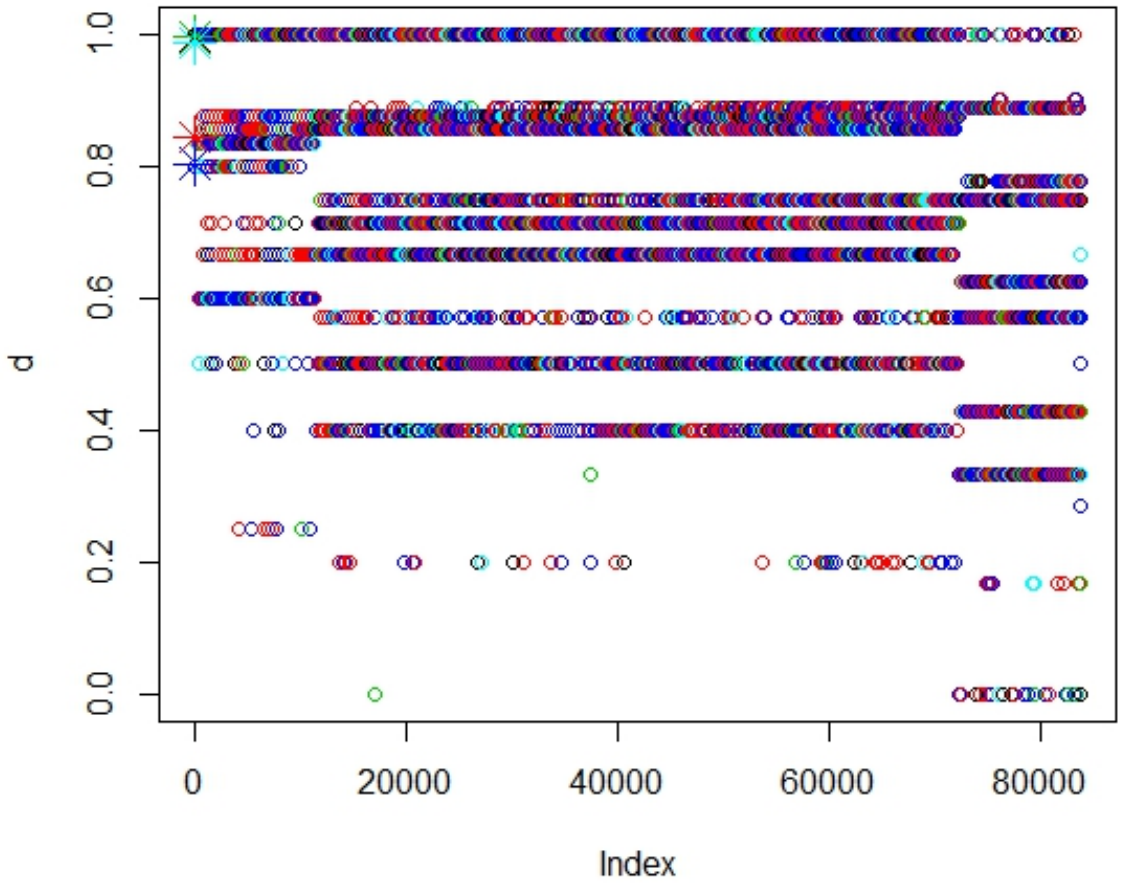


Figure 7.10: Résultat du clustering avec kmeans.