Création d'un package R pour des cartes auto-organisatrices

Laura BENDHAIBA

Stage de fin d'études
18 février - 26 juillet 2013

Table des matières

Remerciements

Mes remerciements s'adressent en premier lieu à mes encadrantes, Nathalie Villa-Vialaneix et Madalina Olteanu, pour leur confiance, le temps et les conseils prodigués tout au long du stage.

Je souhaite également remercier Régis Sabaddin, directeur de l'unité MIA de l'INRA de Toulouse, ainsi que toute son équipe, pour leur accueil fort chaleureux.

Enfin, je remercie également Julien Jacques, mon tuteur école, pour la formation théorique apportée au cours de mes 3 années à Polytech'Lille.

Chapitre 1

Introduction

Le présent rapport de stage a été effectué dans le cadre de ma 5ème et dernière année d'études à Polytech'Lille dans le département Génie Informatique et Statistique. Nécessaire pour valider mon diplôme d'ingénieur, j'ai choisi d'effectuer mon stage de fin d'études pour le compte du laboratoire de recherche « SAMM » de l'Université Paris 1 Panthéon Sorbonne.

Ce stage fait suite à mon projet de fin d'études, réalisé dans les mêmes conditions durant 5 semaines (du 18 février au 22 mars 2013). Le stage porte donc sur les cartes auto-organisatrices, qui sont un moyen alternatif d'effectuer de la classification (non supervisée) de données. Plus précisément, mon stage est dédié à l'implémentation de cette technique dans le logiciel R [1].

R est un logiciel libre de droit qui propose des nombreuses fonctionnalités. Il est néanmoins possible d'installer des suppléments, appelés packages, afin de les étendre. Le but de mes projet et stage de fin d'études est donc de créer un de ces packages. Le stage a conduit à la publication, fin juillet, de la 4ème version du package. Celui-ci possède un espace dédié sur la forge R[1] et a été présenté aux « 2èmes rencontres R »[2], à Lyon [2]. L'article associé est donné en annexe C).

[1] http://sombrero.r-forge.r-project.org/
[2] http://r2013-lyon.sciencesconf.org/

Chapitre 2

Présentation de l'entreprise

Bien que je sois hébergée à l'INRA de Toulouse, mon stage de fin d'études, et auparavant mon projet de fin d'études, sont réalisés pour le compte du laboratoire SAMM.

2.1 Le laboratoire SAMM (EA 4543)

Le laboratoire SAMM[1] est un laboratoire de recherche de l'Université de Paris 1 Panthéon-Sorbonne. Le sigle SAMM signifie : Statistique, Analyse, Modélisation Multidisciplinaire. Le SAMM a été créé au 1er janvier 2010, de la fusion de l'équipe Marin Mersenne (propre à Paris 1) avec l'équipe SAMOS (composante du Centre d'Economie de la Sorbonne). On trouve parmis les membres du SAMM des statisticiens, des probabilistes, des analystes et des informaticiens.

Les domaines de recherche du SAMM étant variés (mathématiques pures, statistique, techniques neuronales, ...), l'équipe est divisé selon 3 axes de recherche :

- Apprentissage statistique et réseaux,
- Equations d'évolution,
- Statistique.

2.2 L'axe apprentissage statistique et réseaux

L'axe apprentissage statistique et réseaux, dont le responsable est Fabrice Rossi, est composé de :

- 1 professeur,
- 6 maîtres de conférences,
- 1 professeur émérite,
- 2 maîtres de conférences honoraires,
- 4 doctorants.

Les thématiques de recherche de l'équipe s'articulent autour de l'apprentissage statistique, de la classification non supervisée, de l'analyse des graphes et réseaux (inférence, classification...) et de données complexes en général, ainsi que des méthodes neuronales et des cartes auto-organisatrices.

[1] http://samm.univ-paris1.fr/

2.3 Modèles neuronaux et cartes de auto-organisatrices

Parmi les projets de recherche de l'axe apprentissage statistique et réseaux, plusieurs sont consacrés aux modèles neuronaux et aux cartes de auto-organisatrices (aussi souvent appelées « cartes de Kohonen »). Mes projet et stage de fin d'études s'inscrivent dans l'étude de cartes auto-organisées pour des données qualitatives, des séries temporelles et, de manière plus générale, des données non euclidiennes décrites par des tableaux de dissimilarités. Ces travaux sont menés par mes encadrantes Nathalie Villa-Vialaneix et Madalina Olteanu, maîtresses de conférences, respectivement à l'Université de Perpignan Via Domitia et à l'Université Paris 1 ainsi que par Marie Cottrell, professeure émérite de l'Université Paris 1, toutes les trois membres du SAMM.

Chapitre 3

Méthodes

Cette section présentera, dans un premier temps, une introduction théorique aux cartes auto-organisatrices. Les références utilisées pour développer le package seront ensuite brièvement décrites puis, pour finir, la section présentera les outils de travail utilisés.

3.1 Les cartes auto-organisatrices

Les cartes auto-organisatrices font partie de la famille des méthodes neuronales, elles-mêmes basées sur l'algorithmique : la solution du problème n'est pas calculée directement mais approchée de manière itérative. Elles permettent de faire de la classification non supervisée de données, tout comme le permettent les k-means ou la classification ascendante hiérarchique, par exemple. Les travaux de référence dans ce domaine sont ceux de Teuvo Kohonen [3]. Mais, à la différence des méthodes de classification ordinaires, les cartes auto-organisatrices organisent les classes trouvées sur une carte dont la topologie cherche à respecter la topologie des données d'origine.

3.1.1 Généralités sur l'algorithme de carte auto-organisatrice

Soient n observations dans $\mathbb{R}^d$ qui forment donc une matrice X de dimension $n \times d$. Le principe de l'algorithme est de projeter les n observations sur une carte de petite dimension (typiquement, 2), composées de U unités ou neurones, $\{1, \ldots, U\}$, et munie d'une distance (entre neurones). La projection $f : x_i \to f(x_i) \in \{1, \ldots, U\}$ est effectuée de manière à ce que les distances entre observations soient aussi bien représentées que possible par les distances entre les neurones auxquelles elles sont affectées sur la carte : cette propriété est appelée *conservation topologique*.

Chacun des U neurones est représenté par un prototype qui est un point de $\mathbb{R}^d$. Les prototypes sont les représentants des neurones (et donc des observations classées dans les neurones) dans l'espace de départ. Les prototypes doivent être initialisés au début de l'algorithme. On peut, par exemple, les initialiser dans l'échantillon de départ ou bien encore les tirer aléatoirement dans $\mathbb{R}^d$.

Le critère de qualité minimisé pour définir la classification f et les prototypes $(p_u)_u$ est appelé l'énergie

$$\mathcal{E} = \sum_{i=1}^{n} \sum_{u=1}^{U} h(f(x_i), u) \|x_i - p_u\|^2 \tag{3.1}$$

où h est une fonction décroissante des distances entre neurones sur la carte : elle pondère l'influence dans l'énergie des distances entre observations et prototypes, selon que l'observation x_i est classée dans un neurone proche de u (poids fort) ou dans un neurone éloigné de u (poids faible). Le critère est similaire à celui de l'algorithme k-means mais ici, toutes les distances entre observations et prototypes sont prises en compte contrairement à l'algorithme k-means où seules les distances entre observations et centroïdes de la classe correspondante sont prises en compte.

3.1.2 Version stochastique de l'algorithme pour données numériques

L'énergie $\mathcal{E}$ de l'équation (3.1) ne peut être minimisée de manière exacte. Il existe deux versions pour approcher sa minimisation : la version batch et la version stochastique. Ici, nous avons étudié la version stochastique de l'algorithme.

La version stochastique de l'algorithme répète deux étapes : une *étape d'affectation* qui, pour une observation sélectionnée aléatoirement dans l'échantillon, va déterminer le neurone de la grille dans laquelle elle doit être classée : le neurone choisi est simplement celui dont le prototype est le plus proche de l'observation. La seconde étape, appelée *étape de représentation*, remet à jour tous les prototypes pour qu'ils aillent dans le sens de la minimisation de l'énergie. Dans la version stochastique de l'algorithme, cette étape est effectuée à l'aide d'une pseudo-descente de gradient stochastique autour de l'observation tirée dans l'étape d'affectation.

Ces étapes sont clairement identifiables dans la séquence algorithmique suivante :

Algorithm 1 On-line numeric SOM

1: Initialisation : choisir aléatoirement p_1^0 ,..., p_U^0 in $\mathbb{R}^d$
2: **for** $t = 1 \to T$ **do**
3: Choisir aléatoirement $i \in \{1, \ldots, n\}$
4: Affecter

$$f^t(x_i) \leftarrow \arg \min_{u=1,\ldots,U} \|x_i - p_u^{t-1}\|_{\mathbb{R}^d}$$

5: **for all** $u = 1 \to U$ **do** Mettre à jour les prototypes
6: $p_u^t \leftarrow p_u^{t-1} + \mu_t h^t(\delta(f^t(x_i), u)) \left(x_i - p_u^{t-1}\right)$
7: **end for**
8: **end for**

3.1.3 Adaptation de l'algorithme pour des données décrites par des tables de contingence

Lorsque l'on dispose de deux variables qualitatives, l'analyse des correspondances classique consiste à faire une analyse en composantes principales pondérée, utilisant la distance du χ^2, simultanément sur les profils lignes et les profils colonnes. Ce principe est adapté pour aboutir à une déclinaison de l'algorithme de carte organisatrice pour les tables de contingence qui est appelée « Korresp » [4].

Les tables de contingence ici considérées possèdent au plus 2 variables qualitatives, ayant respectivement p et q modalités. La matrice contient donc p lignes et q colonnes. Le terme n_{ij} est le nombre d'individus appartenant à la fois à la classe i de la première variable et à la classe j de la seconde variable.

Avant d'appliquer l'algorithme, la table de contingence doit subir un pré-traitement qui calcule les profils lignes (PL) et les profils colonnes (PC). La matrice d'entrée est ensuite obtenue en juxtaposant les matrices de fréquences comme indiqué dans l'équation (3.2).

$$X = \begin{pmatrix} PL & 0 \\ 0 & PC \end{pmatrix} \tag{3.2}$$

Dans le cas Korresp, l'algorithme de carte auto-organisatrice présenté dans l'algorithme 1 est appliqué aux données de l'équation (3.2) mais l'étape d'affectation est légèrement modifiée. Les bornes de l'intervalle dans lequel est choisie aléatoirement l'observation sont successivement :

- $1 \ldots p$: on considère un profil ligne
- $p+1 \ldots p+q$: on considère un profil colonne

et l'affectation n'est effectuée que sur la base des coordonnées non nulles de la ligne considérée. Cette alternance permet de classer alternativement les modalités des variables lignes et celles des variables colonnes.

Les prototypes sont de plus remaniés : lorsqu'on s'intéresse à un profil ligne (resp. colonne), on colle les p premières cellules de la ligne tirée (resp. les q dernières) avec les q (resp. p) cellules du profil colonne (resp. ligne) le plus probable ; cette extension des prototypes n'est pas utilisée pour la phase d'affectation mais seulement pour la phase de représentation.

Algorithm 2 On-line korresp SOM

Initialisation : choisir aléatoirement p_1^0, ..., p_U^0 in $\mathbb{R}^{p+q}$
for $t = 1 \to T/2$ **do**
  Profil ligne : Choisir aléatoirement $i \in \{1, \ldots, p\}$
  Affecter

$$f^t(x_i) \leftarrow \arg \min_{u=1,\ldots,U} \|x_{i,1\ldots p} - p^{t-1}_{u,1\ldots p}\|_{\mathbb{R}^p}$$

  for all $j = 1 \to M$ **do** Mettre à jour les prototypes
    $p_u^t \leftarrow p_u^{t-1} + \mu_t h^t(\delta(f^t(x_i), u))\left(x_i^L - p_u^{t-1}\right)$, avec $x_i^L = (x_{i,1\ldots p}, x_{j^*(i),q+1,\ldots,p+q})$ où $j^*(i) = \arg\max_{j=1,\ldots,q} x_{ij}$
  end for
  Profil colonne : Choisir aléatoirement $i \in \{p+1, \ldots, p+q\}$
  Affecter

$$f^t(x_i) \leftarrow \arg \min_{u=1,\ldots,U} \|x_{i,p+1\ldots p+q} - p^{t-1}_{u,p+1\ldots p+q}\|_{\mathbb{R}^q}$$

  for all $j = 1 \to M$ **do** Mettre à jour les prototypes
    $p_j^t \leftarrow p_u^{t-1} + \mu_t h^t(\delta(f^t(x_i), u))\left(x_i^C - p_u^{t-1}\right)$, avec $x_i^C = (x_{j^*(i),1\ldots p}, x_{i,q+1,\ldots,p+q})$ où $j^*(i) = \arg\max_{j=p+1,\ldots,p+q} x_{ij}$
  end for
end for

3.1.4 Adaptation de l'algorithme pour des données décrites par des dissimilarités

Dans de nombreuses applications réelles, il est impossible de décrire les données par des variables numériques. Une solution possible pour contourner ce problème est de les décrire par des mesures de ressemblance (similarité ou dissimilarité). Comme beaucoup d'autres méthodes de data mining, l'algorithme de cartes auto-organisatrices possède une version qui permet de traiter ces données.

Lorsque les données sont décrites par une matrice de dissimilarités, $\mathbf{\Delta}$, il est alors possible d'exprimer les prototypes comme une combinaison convexe symbolique des observations, $p_u \sim \sum_{i=1}^n \beta_{ui} x_i$ avec $\beta_{ui} \geq 0$ et $\sum_i \beta_{ui} = 1$. Cette alternative, appelée relationelle, est celle qui donne les résultats les plus proches de la méthode Euclidienne du cas numérique [5].

L'adaptation relationnelle est proposée par l'algorithme 3.

Algorithm 3 On-line relational SOM

Pour tout $u = 1, \ldots, U$ et $i = 1, \ldots, n$, initialiser aléatoirement β_{ui}^0 dans $\mathbb{R}$, tel que $\beta_{ui}^0 \geq 0$ et $\sum_i^n \beta_{ui}^0 = 1$.
for $t = 1 \to T$ **do**
  Choisir aléatoirement $i \in \{1, \ldots, n\}$
  Affectation : trouver le prototype le plus proche

$$f^t(x_i) \leftarrow \arg \min_{u=1,\ldots,U} \left(\beta_u^{t-1}\mathbf{\Delta}\right)_i - \frac{1}{2}(\beta_u^{t-1})^T \mathbf{\Delta} \beta_u^{t-1}$$

  Représentation : $\forall\, u = 1, \ldots, U$,

$$\beta_u^t \leftarrow \beta_u^{t-1} + \mu_t h^t(d(f^t(x_i), u))\left(\mathbf{1}_i - \beta_u^{t-1}\right)$$

  où $\mathbf{1}_i$ est un vecteur dont le seul coefficient non nul, à la *ième* position, est égal à 1.
end for

Dans le cas de données relationnelles, les calculs de distances sont modifiés. Les formules 3.3 et 3.4 expriment respectivement le calcul de la distance d'une observation x_i à un prototype p_u et le calcul de la distance entre deux prototypes p_u et $p_{u'}$.

$$(\mathbf{\Delta}\beta_u)_i - \frac{1}{2}\beta_u^T \mathbf{\Delta}\beta_u \tag{3.3}$$

$$-\frac{1}{2}(\beta_u - \beta_{u'})^T \mathbf{\Delta}(\beta_u - \beta_{u'}) \tag{3.4}$$

Ces calculs servent notamment à calculer la qualité de la projection et sont implémentés dans la fonction `calculateProtoDist`.

3.2 L'existant

Le but de mon travail est de proposer et de tester une implémentation de la version stochastique de l'algorithme de cartes auto-organisatrices pour le logiciel libre R[1] [1], qui puisse traiter des données numériques et non numériques, comme décrit dans les sections précédentes.

Il existe déjà des packages de ce type, comme le package **yasomi**[2] développé par Fabrice Rossi en 2011, qui traite de cartes auto-organisatrices et implémente la version BATCH de l'algorithme pour des données numériques ou des données décrites par des dissimilarités. Il peut être installé à l'aide de la commande :

```
install.packages("yasomi", repos="http://R-Forge.R-project.org").
```

Egalement, les scripts SAS developpés par Patrick Letremy [3] ont été un point de repère concernant les modifications à apporter à l'algorithme pour qu'il puisse traiter des données sous forme de tableaux de contingence.

3.3 Les outils de travail

3.3.1 Git

Afin de faciliter le suivi des évolutions de mes travaux, j'ai été amenée à utiliser le logiciel Git[4]. Ce logiciel permet, via la ligne de commande, de gérer des versions de fichiers. Git mémorise toutes les sauvegardes, créant un historique permettant de suivre les évolutions. Utilisable avec un serveur distant, Git permet également de synchroniser des répertoires, ce qui rend possible une utilisation multi-utilisateurs. Si plusieurs personnes modifient le même document, Git tente de gérer le conflit. Si les modifications portent sur des lignes différentes, le conflit est réglé automatique par fusion des deux fichiers. Sinon, Git indique à l'utilisateur les endroits où se trouvent les conflits afin qu'il les résolve.

Typiquement, voici le processus de travail avec Git :

- l'utilisateur modifie ou crée un fichier,
- quand elles sont terminées, il effectue un `git commit`, décrit par un `commit message`,
- et enfin envoie ces modifications sur le serveur avec un `git push`,
- les autres utilisateurs peuvent récupérer les modifications grâce à un `git pull`.

D'autres instructions existent, parmi lesquelles :

- `amend`, pour modifier le commit précédent ;
- `git reset`, pour annuler le commit précédent ;
- `git tag`, pour ajouter un tag à un commit, ce qui le rend plus facilement identifiable (par défaut, un commit est identifié par un SHA de 40 caractères ;

[1] http://www.r-project.org/
[2] « yasomi » pour Yet Another Self Organising Map Implementation
[3] http://samos.univ-paris1.fr/Programmes-bases-sur-l-algorithme
[4] http://git-scm.com/

- `git checkout` pour changer de branche ; pour **SOMbrero**, 2 branches ont été utilisées : une branche master, pour développer la version 0.4 du package, et une branche dev, sur laquelle étaient implémentées des modifications qui ne seront disponibles que dans la version 0.5 ;
- `git cherry-pick`, qui permet de sélectionner un commit d'une branche et de l'intégrer à une autre branche.

Une visualisation, via le logiciel Gource[5], de l'évolution de mon dépôt Git est disponible à http://youtu.be/sZSaIf6in3g.

3.3.2 R Studio

Pour développer les scripts R j'ai utilisé le logiciel libre RStudio[6], qui est un environnement de développement intégré dédié à R. Il permet d'avoir sur le même écran (voir figure 3.1) :

- les scripts,
- la console,
- l'environnement de travail, l'historique (et Git, avec lequel il est compatible),
- et d'avoir accès aux figures, packages et aides en ligne.

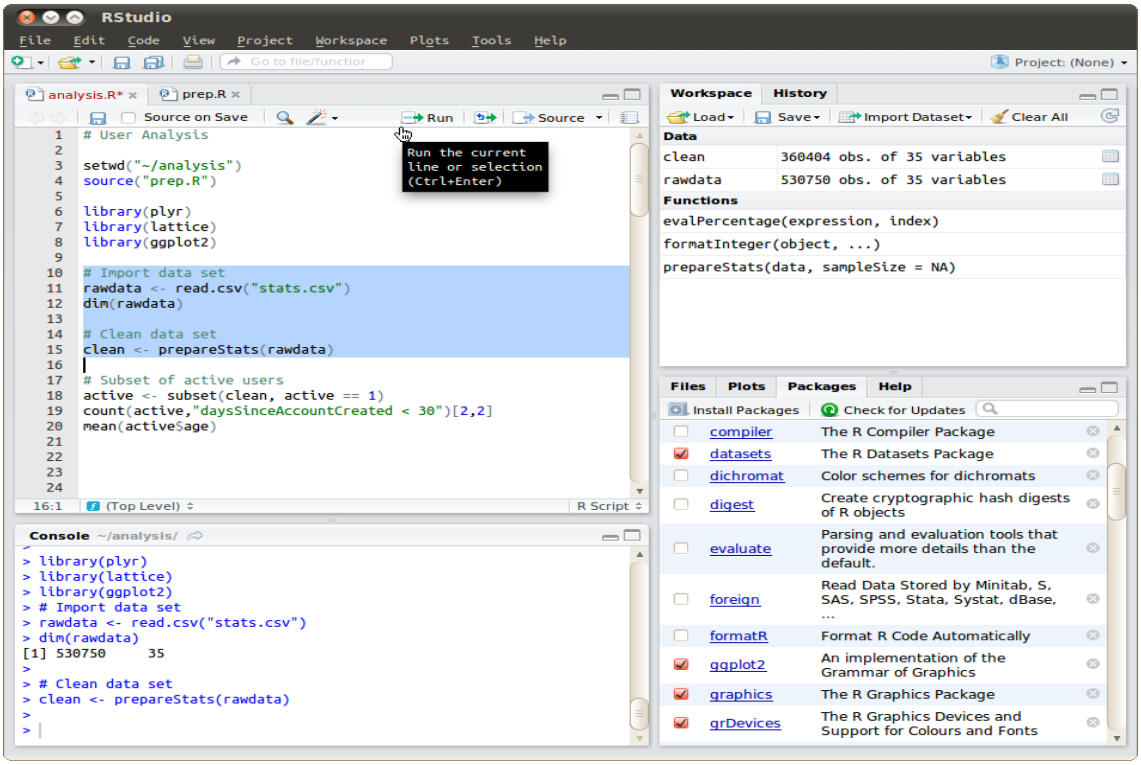


FIGURE 3.1 – Saisie d'écran de R Studio, source : http://www.rstudio.com

[5] http://code.google.com/p/gource/
[6] http://www.rstudio.com/

Chapitre 4

Travail effectué

La version 0.4 du package est constituée de 7 scripts R contenant, au total, 53 fonctions, dont 21 sont visibles par l'utilisateur. Cette section détaille les scripts écrits, la structure propre à un package R et présente des études de cas.

4.1 Description des programmes

Le code permettant de faire tourner l'algorithme est segmenté en 7 fichiers, comme suit :

- 1 fichier d'initialisation, éxécuté lors de l'installation du package par l'utilisateur (fichier *init.R*),
- 3 fichiers fonctionnant de manière inclusive pour définir, en premier lieu la grille (fichier *grids.R*), puis les paramètres de l'algorithme (*parameters.R*) et enfin exécuter celui-ci (*som.R*),
- 1 fichier qui traite des différents graphiques (*plots.R*),
- 1 fichier dédié au calcul de la qualité de la projection (*quality.R*),
- 1 fichier qui réalise une CAH sur la classification produite par l'algorithme de cartes auto-organisatrices (*superclasses.R*).

Le graphique de la figure 4.1 affiche les dépendances entre fonctions du package.

Le système d'objets S3 a été utilisé dans ce package. Pour chacune des classes d'objets créées, les principales méthodes génériques ont été redéfinies, en respectant les conventions de nommage[1].

Ne seront détaillés dans les sections suivantes que les fichiers et fonctionnalités apportées au package après la remise du rapport de projet.

4.1.1 Fichier *som.R*

Suite aux ajouts de traitements portant sur des données non numériques, la structure de ce script a été modifiée : la fonction `trainSOM` est désormais celle qui contient les traitements, et qui appelle des sous-fonctions invisibles à l'utilisateur lors des différentes étapes de l'algorithme.

Le code du script est disponible en annexe A.1.

4.1.2 Fichier *plots.R*

Le fichier *plots.R* implémente les différents graphiques qui peuvent être produits à partir d'un object de classe `somRes`. Il s'agit donc en fait de la méthode S3 `plot.somRes`.

La fonction `plot.somRes` possède 2 arguments qui permettent de définir le graphique à produire. Le premier, `what`, est à choisir parmi 4 possibilités : `obs`, `prototypes`, `energy` et `add`. Le second, `type`, permet de préciser, pour chacune des modalités de `what`, le graphique voulu.

[1] La fonction `toto()` pour l'object de classe `maClasse` sera nommée `toto.maClasse()`.

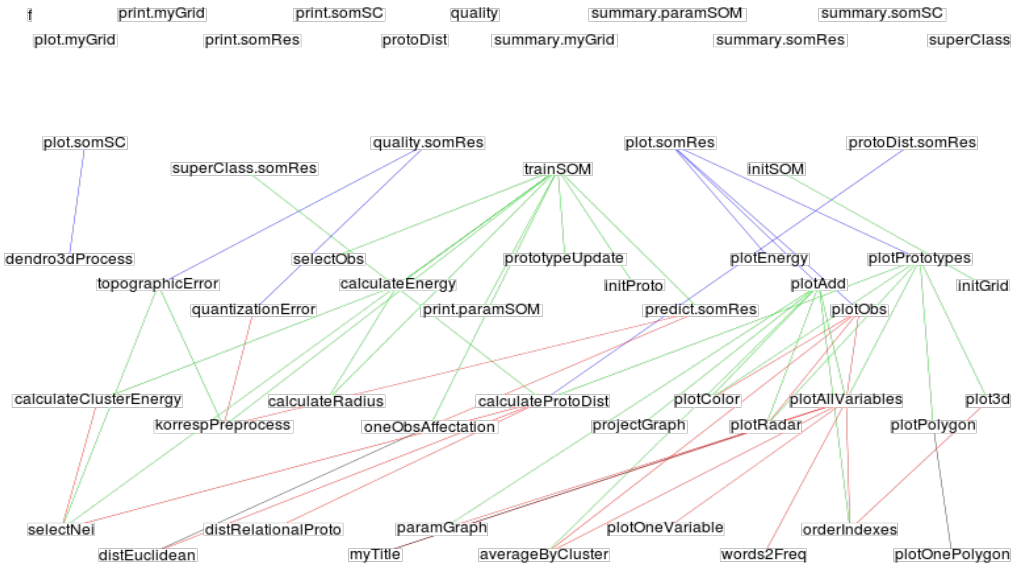


FIGURE 4.1 – Dépendances entre les fonctions dans **SOMbrero** 0.4

La tableau de la figure 4.2 résume tous les graphiques disponibles en détaillant les combinaisons des paramètres `what` et `type` correspondantes et les cas dans lesquels chaque graphique est disponible.

Le code complet est disponible en annexe A.2.

Graphiques de valeurs relatifs aux observations et/ou aux prototypes

De nouveaux graphiques représentant le niveau de valeur des prototypes et/ou des observations ont été implémentés (fonctionnement similaire au graphique `color`) :

- `lines`, qui représente le niveau de toutes les variables du jeu de données avec des lignes (voir la figure 4.18 pour un exemple),
- `barplot`, idem en histogrammes à barres verticales (figure 4.24),
- `radar`, idem sous forme de diagrammes circulaires (figure 4.5),
- `boxplot`, idem (mais avec une limitations à 5 variables numériques pour conserver la lisibilité du graphique) mais avec des boîtes à moustaches (figure 4.12),
- `3d`, qui représente sur un graphique 3 dimensions le niveau d'une des variables qui sera précisée à l'argument `variable` (voir figure 4.15).

Dans le cas `korresp`, les graphiques relatifs aux prototypes identifiés par `x2` sur la figure 4.2, possèdent un argument supplémentaire, `view`, qui permet d'indiquer si le graphique doit porter sur les variables lignes ou sur les variables colonnes (voir les figures 4.18 et 4.19 pour illustration).

Graphiques de distances entre les prototypes

Les graphiques présentés dans cette section utilisent les distances calculées soit entre une observation et un prototype, soit entre deux prototypes. Le calcul des distances étant différent selon le type de donnée, une fonction `calculateProtoDist` a été écrite.

type	numeric					korresp					relational				
	what				SC	what				SC	what				SC
	energy	obs	proto	add		energy	obs	proto	add		energy	obs	proto	add	
no type	x					x					x				
hitmap		x			x		x			x		x			x
color		x	x	x	x2			x2		x2				x	
lines		x	x	x	x2			x2		x2			x	x	x2
barplot		x	x	x	x2			x					x	x	x2
radar		x	x	x	x2			x					x	x	x2
pie				x	x									x	x2
boxplot		x		x	x2									x	
3d			x					x2							
poly.dist			x		x			x		x			x		x
umatrix			x					x					x		
smooth.dist			x					x					x		
words				x										x	
names		x		x			x*					x		x	
graph				x	x									x	x
mds			x		x			x		x			x		x
grid.dist			x					x					x		
grid					x					x					x
dendrogram					x					x					x
dendro3d					x					x					x

FIGURE 4.2 – Récapitulatif des graphiques implémentés dans **SOMbrero** 0.4

Le package **SOMbrero** propose 5 graphiques afin de visualiser les distances entre prototypes :

- `poly.dist` trace à l'intérieur de chaque neurone un polygone dont l'éloignement du sommet au coin de la case est proportionnel à la distance entre les 2 prototypes concernés (voir la figure 4.7 pour un exemple),
- `umatrix` représente la moyenne des distances d'un neurone à tous les autres avec une plage de couleur (figure 4.17),
- `smooth.dist` a le même comportement qu'`umatrix` mais avec des couleurs lissées (voir figure 4.16),
- `mds` projette dans un plan à 2 dimensions les prototypes (cf figure 4.8),
- `grid.dist` trace pour chaque couple de prototypes la distance calculée en fonction de la distance sur la grille.

Graphique détaillant la répartition des observations

Le graphique `names` permet de visualiser en détail la répartition des observations sur la carte. Grâce à l'appel de la fonction `wordcloud` du package homonyme, le nom des observations (nom des lignes du jeu de données passé en entrée) est imprimé dans le neurone sous laquelle elle est classée sous forme de nuage de mots. Un exemple est donné en section 4.3, figure 4.6.

Pour le cas `korresp`, le graphique `names` est signalé par une `*` sur la figure 4.2 : du fait du traitement appliqué aux données dans ce cas, les noms des variables lignes et colonnes seront imprimés sur la carte (voir la figure 4.14 pour un exemple).

Graphiques relatifs à une variable additionnelle

De nombreux graphiques sont implémentés pour considérer une variable additionnelle, qui ne fait donc pas partie du jeu de données initial. La variable additionnelle peut être :

- un vecteur numérique : le graphique `color` est alors disponible, la plage de couleur étant appliquée sur les valeurs de la variable additionnelle ;
- une matrice numérique : les graphiques `lines`, `barplot`, `radar` et `boxplot`, définis plus haut, sont alors disponibles ;
- un vecteur non numérique, ce qui permet alors de produire soit un graphique de diagrammes circulaires suivant la répartition de la variable additionnelle (graphique `pie`, voir figure 4.9), soit un graphique nuage de mots (graphique `names`, voir figure 4.6 pour un exemple).

4.1.3 Fichier *superclasses.R*

Etant donné le nombre important de classes produites par l'algorithme de cartes auto-organisatrices, il est courant d'appliquer une classification ascendante hiérarchique sur ses résultats.

Le script est disponible en annexe A.4.

La fonction `superClass`

La fonction `superClass`, qui retourne un object de classe `somSC`, prend en argument un objet de classe `somRes`. Peuvent éventuellement lui être passés :

- l'argument `method` de la fonction `hclust`,
- l'argument `members` de la fonction `hclust`,
- l'argument `k` ou l'argument `h` utilisés par la fonction `cutree`.

Si ni `k` ni `h` ne sont précisés, le nombre de super classes n'est alors pas défini mais il est tout de même possible de visualiser le dendrogramme en appelant la fonction `plot.somSC`. Si l'un des 2 arguments est fourni, le découpage sera fait soit en k classes, soit à hauteur h.

Graphiques

Puisque qu'un nouveau type d'objet est créé par la fonction `superClass`, ses fonctions usuelles ont été défini en utilisant l'orientation objet S3 de R.

La fonction `plot.somSC` propose 3 graphiques propres aux super classes : un dendrogramme (`type="dendro"`, exemple en figure 4.10), un dendrogramme en 3 dimensions (`type="dendro3d"`, figure 4.25) et une visualisation des groupes sur la grille (`type="grid"`, figure 4.11). Elle permet également de produire certains des graphiques de `plot.somRes` en identifiant les super classes soit avec du texte, soit avec des couleurs. Pour le détail des graphiques disponibles, voir la figure 4.2. Les cellules contenant `x2` identifient les graphiques pour lesquels il est possible d'utiliser soit une variable du jeu de données soit une variable additionnelle (dans ce cas, l'argument `add.type` doit être instancié à TRUE).

4.1.4 Fichier *quality.R*

Comme évoqué en section 3.1, les cartes auto organisatrices respectent la topologie des données d'origine. Parmi les différents indicateurs de qualité, nous en avons selectionné deux. Ils permettent de mesurer la qualité de la projection des données sur la carte. Ces critères de qualité sont calculés par ce script. Par défaut, la fonction retourne les deux critères, mais le critère à calculer peut être spécifié à l'argument `quality.type`.

Le premier critère est appelé *erreur topographique* : pour le calcul de ce critère, le second neurone le plus proche de chaque observation est déterminé et la fréquence du nombre de fois où ce second neurone ne se trouve pas dans un voisinage à distance un sur la grille du neurone d'affectation est déterminée. Cet

indicateur est donc un nombre compris entre 0 (projection de bonne qualité) et 1 (projection de mauvaise qualité). Sa valeur est donnée dans l'équation (4.1) :

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}_{\{f(x_i)\notin\mathcal{N}^1(f^{(2)}(x_i))\}} \tag{4.1}$$

où $f^{(2)}(x_i) = \arg\min_{u\neq f(x_i)}\delta(x_i, p_u)$, δ représentant la distance entre x_i et p_u qui dépend de la variante de l'algorithme qui est utilisée, et $\mathcal{N}^1(u)$ est le voisinage à distance 1, sur la grille, du neurone u.

Le second crtière, l'*erreur de quantification*, est la moyenne de la distance entre les observations et les prototypes de leurs classes. Dans le cas particulier où le rayon final de la carte est 0 (réduit au neurone lui-même), les prototypes des classes en sont les centroïdes et cette erreur est la variance intra-classes. C'est donc un nombre positif dont la valeur doit être proche de 0 pour une bonne qualité de projection. Elle est explicitée dans l'équation (4.2),

$$\frac{1}{n}\sum_{i=1}^{n}\delta(x_i, p_{f(x_i)}) \tag{4.2}$$

Une illustration est donnée en section 4.3, le code est lui disponible en annexe A.3.

4.1.5 Résumé des classes et méthodes implémentées

En résumé, l'utilisateur a à sa disposition les classes et fonctions associées suivantes :

- `myGrid`, un objet grille et ses fonctions :
  - `print.myGrid`, qui imprime les caractéristiques de la grille,
  - `plot.myGrid`, qui produit le graphique correspond à la grille,
  - `summary.myGrid`, qui imprime la classe puis l'objet lui-même.
- `paramSOM`, un objet-liste regroupant les paramètres d'entrée de l'algorithme, pour lequel il existe les fonctions :
  - `print.paramSOM`, qui affiche la liste des valeurs des paramètres,
  - `summary.paramSOM`, qui affiche la classe puis l'objet lui-même.
- `somRes`, l'objet-résultat de l'algorithme à partir duquel on peut appeler :
  - `print.somRes`, qui imprime quelques informations sur les éléments contenus par l'objet,
  - `plot.somRes`, qui permet de produire de nombreux graphiques,
  - `summary.somRes`, qui affiche un résumé de l'objet, notamment une analyse de variance dans les cas `numeric` et `relational`
  - `predict.somRes`, qui prédit le classement d'une nouvelle observation (voir exemple en section 4.3),
  - `quality.somRes`, qui calcule des indicateurs mesurant la qualité de la projection des données sur la carte,
  - `superClass.somRes`, qui applique une classification ascendante hiérarchique,
  - `protoDist.somRes`, qui permet de calculer les distances, selon le type de données.
- `somSC`, l'objet-résultat retourné après appel de la fonction `superClass` sur un objet `somRes`, pour lequel sont définis :
  - `print.somSC`, qui affiche les principales caractéristiques de l'objet,
  - `plot.somSC`, qui permet la production de graphiques en considérant les informations apportées par l'objet,
  - `summary.somSC`, qui affiche l'objet, sa table de fréquences et ses valeurs.

4.2 Création d'un package R

Un package R est constitué d'un ensemble de fichiers et de répertoires dont la nomenclature est normalisée selon les directives publiées par le CRAN[2] (voir aussi [6] pour une version simplifiée en français). Le répertoire racine doit porter le nom du package. Voici l'arborescence du package **SOMbrero**, version 0.4 :

- un répertoire */R/*, qui contient les scripts R,
- un répertoire */man/*, qui contient les fichiers d'aide (fichiers d'extension `.Rd`, chaque fonction publique doit être référencée dans un fichier d'aide),
- un répertoire */data/*, qui contient 2 jeux de données (cf section 4.2.1),
- un répertoire */tests/*, qui contient des tests bloquants qui arrêtent la compilation du package en cas d'erreur lors de leur exécution (code donné en annexe A.5),
- un réperstoire */demo/*, qui contient des démonstration exécutées automatiquement lorsque l'utilisateur lance la commande `demo()`,
- un répertoire */inst/doc/*, qui contient des documentations utilisateurs, aussi appelées *vignettes* : ces documentations ont été réalisées avec Markdown et compilé avec le package **knitr** comme autorisé depuis la version 3.0 du logiciel,
- un fichier *NAMESPACE*, qui précise la visibilité des fonctions,
- un fichier *DESCRIPTION*,
- un fichier *NEWS*, qui permet retrace l'historique des versions du package,
- un fichier *TODO*, qui répertorie les fonctionnalités à implémenter ou les problèmes à corriger.

4.2.1 Jeux de données

Deux jeux de données sont automatiquement chargés lors du téléchargement du package **SOMbrero** par l'utilisateur :

- *presidentielles2002*, un jeu de données donnant, pour chaque department français, les résultats des 16 candidats au premier tour de l'élection présidentielles de 2002,
- *lesmis*, un graphe de co-apparition des personnages du roman Les Misérables de Victor Hugo ; la matrice de distances entre les personnages, calculée par la fonction `shortest.path` du package **igraph**, est stockée dans l'objet *dissim.lesmis*.

Ces jeux de données sont utilisés lors des sections *examples* des documentations R et sont détaillés dans les fichiers de documentations utilisateur. Ils seront brièvement traités dans la section 4.3 du présent rapport.

4.2.2 Documentations

Les deux types de documentations évoquées dans mon précédent rapport de projet ont été developpées. Pour exemples, le code de la documentation R de la fonction `quality` est donné en annexe B.1 ; la documentation utilisateur complète correspondant à la version korresp de l'algorithme est donnée en annexe B.2.

[2] http://cran.r-project.org/doc/manuals/R-exts.html

4.2.3 Démos

Le package **SOMbrero** comporte 4 démos :

- une démo utilisant des données numériques,
- une démo utilisant une matrice de contingence,
- une démo utilisant une matrice de dissimilarité,
- et une démo produisant le logo du package.

Elles sont accessible par la commande `demo(package="SOMbrero")` dans R. Le code de la 4ème démo est donné en annexe A.6.

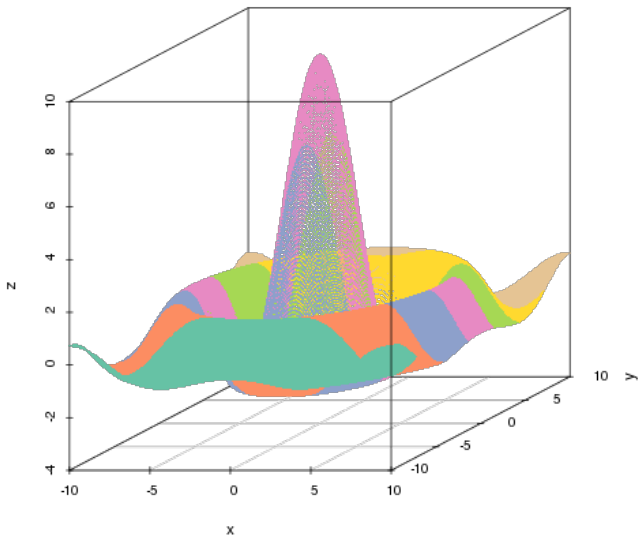


FIGURE 4.3 – Le logo du package **SOMbrero**

4.2.4 Test

Les fichiers du répertoire */tests/* sont exécutés lors de la compilation du package. **SOMbrero** ne comporte qu'un seul fichier test (dont le code est donné en annexe A.5), qui effectue des vérifications sur les valeurs des prototypes pour l'algorithme relationnel (les prototypes doivent être des combinaisons convexes des variables de départ). Si les conditions ne sont pas remplies, la compilation du package échoue.

4.2.5 Versions

Comme mentionné plus haut, le présent rapport fait état de la 4ème version du package.

La 1ère version, finalisée pour la soutenance de mon projet de fin d'études, a été mise au point fin mars. Celle-ci ne considérait que le cas numérique et n'implémentait que très peu de fonctionnalités.

La 2ème version, parue le 18 avril (avec une mise à jour le 20 mai), proposait le traitement complet du cas numérique.

La 3ème version ajoutait la possibilité de traiter des tableaux de contingence.

La 4ème et dernière version à ce jour, offre, en plus, la possibilité de traiter des matrices de dissimilarités.

La figure 4.4 présente l'évolution des différentes versions du package au cours du temps.

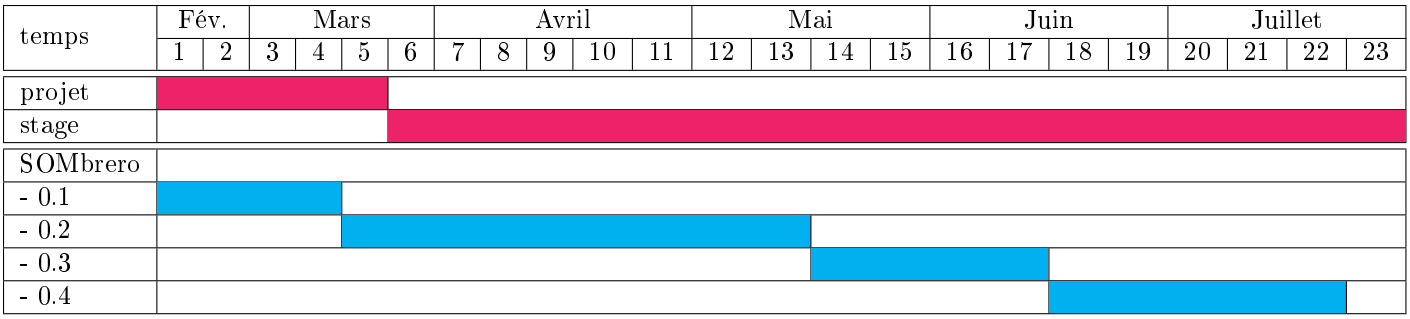


FIGURE 4.4 –

Au terme du stage, le package possède, comme cela a été dit en introduction, une page dédiée sur R-Forge[3] et a été présenté aux « 2èmes rencontres R »[4], à Lyon [2].

4.3 Études de cas

Cette section va permettre d'illustrer le fonctionnement de l'algorithme implémenté via trois études de cas correspondant chacune à un type de données. L'exemple numérique utilisera un jeu de données classique de R à savoir les données `iris`. L'exemple pour les matrices de contingence sera présenté avec des données émanant de l'élection présidentielle de 2002. Enfin, l'algorithme pour tableaux de dissimilarité sera illustré par une matrice de dissimilarités calculée à partir d'un graphe relatif aux personnages du roman Les Misérables de Victor Hugo.

4.3.1 Illustration numérique : les données *iris*

Les données `iris` disponibles dans R sont des données célèbres, introduites par Sir Ronald Fisher en 1936 pour illustrer l'analyse factorielle discriminante. Ce jeu de données contient initialement 5 variables, décrivant des fleurs d'iris issues de 3 espèces. Les variables sont : la longueur du pétale, la largeur du pétale, la longeur du sépale, la largeur du sépale et l'espèce d'iris. Dans les exemples donnés ci-dessous seulement les quatre premières variables du jeu de données sont exploitées (variables numériques). L'algorithme est utilisé par la commande :

```
iris.som <- trainSOM(iris[,1:4])
```

Cette section ne présentera que les fonctionnalités ajoutées au package après le rendu du rapport de projet.

La fonction `predict.somRes`, ajoutée au package dès la version 0.2, permet de prédire l'affectation d'une nouvelle observation. L'exemple suivant considère comme nouvelle observation la première observation du jeu de données `iris`.

```
predict(iris.som, iris[1, 1:4])
##  1
## 15
iris.som$clustering[1]
##  1
## 15
```

La fonction prédit que la nouvelle observation sera affectée au neurone 1, ce qui est cohérent avec le résultat de la classification où cette même observation est effectivement affectée au neurone 1.

La graphique `barplot`, tout comme le graphique `color` déjà présenté dans le rapport de projet, permet de visualiser le profil des observations de chaque neurone. La figure 4.5 indique donc, via la longueur de chaque part du diagramme circulaire, le niveau de valeur des observations pour chacune des 4 variables numériques.

[3] http://sombrero.r-forge.r-project.org/
[4] http://r2013-lyon.sciencesconf.org/

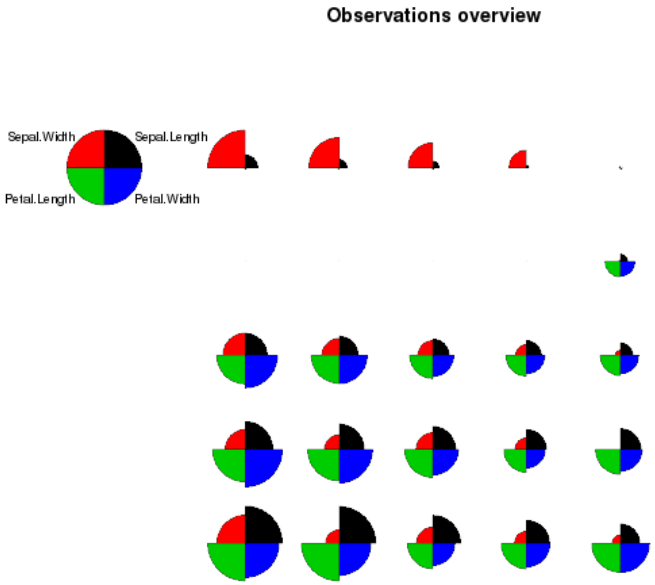


FIGURE 4.5 – Niveau de valeur des prototypes

Afin de permettre une visualisation détaillée de la répartition des observations sur la carte, le graphique `names` imprime, dans chaque neurone, le nom des observations qui y sont affectées. La figure 4.6 permet donc de localiser les différentes fleurs sur la carte. Les fleurs sont indiquées par leur numéro de ligne. Compte tenu du nombre d'observations dans un neurone et de la taille de l'image, un avertissement indiquant que certains noms n'ont pu être affichés peut apparaître lors de l'éxécution. Ce graphique étant relatif aux observations, les 2 neurones vides sont des neurones non utilisés dans la classification. En revanche, ces neurones possèdent des prototypes ; c'est pourquoi, les graphiques relatifs aux prototypes n'ont jamais de neurones vides.

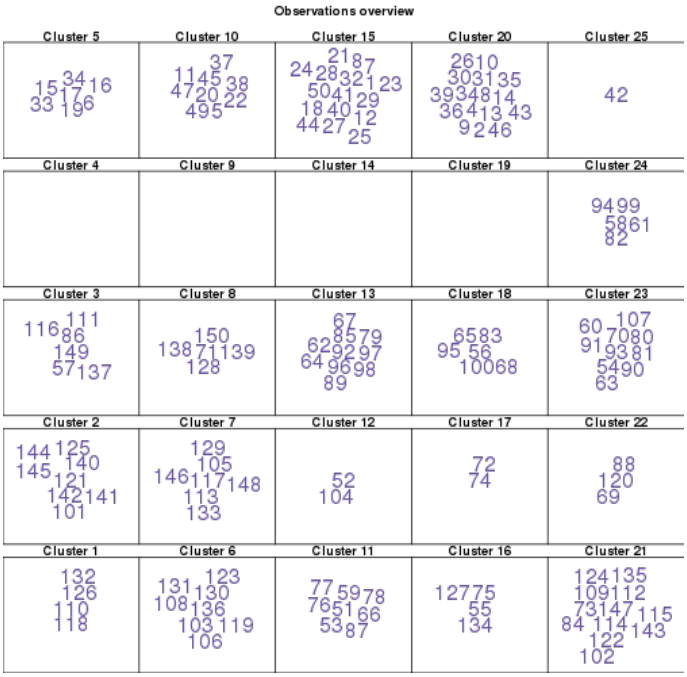


FIGURE 4.6 – Répartition des observations sur la carte

Les distances entre prototypes sont utilisées pour analyser la proximité ou au contraire l'éloignement de deux neurones voisins. Le graphique `poly.dist` représente ces distances par des polygones dont la distance au sommet est proportionnelle. Le code couleur transcrit le nombre d'observations dans le neurone. Les 5 neurones du haut de la carte semblent, d'après la figure 4.7 être différents des autres.

FIGURE 4.7 – Représentation des distances entre prototypes

La projection multi-dimensionnelle des prototypes (graphique appelé `mds`) sur un plan à 2 dimensions permet également de visualiser les distances. On retrouve les 5 neurones cités ci-dessus à l'extrême droite de la figure 4.8.

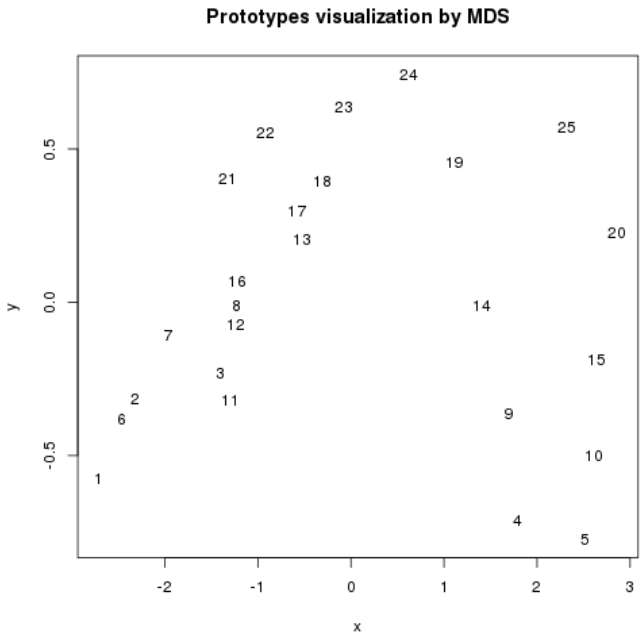


FIGURE 4.8 – Visualisation des prototypes projetés

Comme détaillé dans la section 4.1.2, il est possible de produire des graphiques en considérant une variable qui n'appartient pas au jeu de données initial. C'est le cas de la figure 4.9, produite à partir du graphique `pie`, pour laquelle la variable additionnelle n'est autre que la cinquième variable du jeu de données `iris` qui code l'espèce de la fleur. Cette variable a trois modalités : setosa, virginica et versicolor. Le graphique de type `pie` présenté ici représente, pour chaque neurone, un diagramme circulaire donnant la répartition des espèces au sein du neurone. On voit ici que la majorité des neurones ne regroupent qu'une seule espèce.

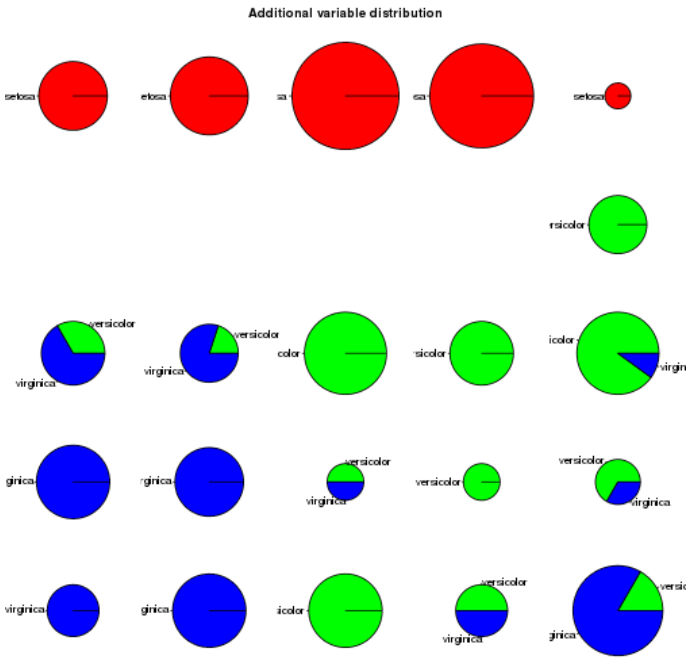


FIGURE 4.9 – Répartition des espèces de fleurs

La fonction de qualité, détaillée en section 4.1.4, est appelée de la manière suivante :

```
quality(iris.som)
```

Et fourni par défaut comme résultat :

```
## $topographic
## [1] 0.06
##
## $quantization
## [1] 0.2244
```

Dans notre cas, l'erreur topographique est égale à 0.06 ce qui traduit une projection de bonne qualité puisqu'il y a seulement 9[5] cas pour lesquels le second voisin n'est pas voisin du premier.

Comme défini en section 4.1.3, la fonction `superClass` permet d'effectuer une classification ascendante hiérarchique. L'instruction suivante permet d'effectuer la CAH en conservant 3 classes :

```
iris.sc <- superClass(iris.som, k=3)
```

Grâce aux commandes :

```
plot(iris.sc, plot.var = FALSE)
plot(iris.sc, type = "grid", plot.legend = TRUE)
plot(iris.sc, type="boxplot", print.title=TRUE)
plot(iris.sc, type = "mds", plot.legend = TRUE, cex = 2)
```

on obtient respectivement les figures 4.10, 4.11 et 4.13. Comme détaillé en section 4.1.3 :

- la figure 4.10 produit le dendrogramme identifiant les super classes obtenues ;
- la figure 4.11 identifie les super classes en couleurs sur la grille ;
- la figure 4.12 petmet de visualiser, comme le graphique 4.5, le profil des observations de chaque neurone, l'information de la super classe étant apportée par un code couleur ;

[5] $0.06 * nrow(iris) = 0.06 * 150 = 9$

- la figure 4.13 est identique à la figure 4.8 mais apporte l'information des super classes avec des couleurs.

On voit sur ces trois figures que les 5 neurones identifiés comme différents plus haut sont regroupés dans le troisième super cluster.

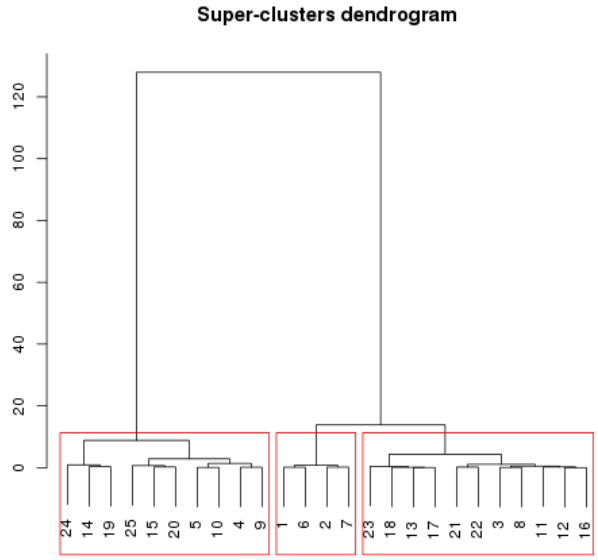


FIGURE 4.10 – Dendrogramme des super classes obtenues par CAH

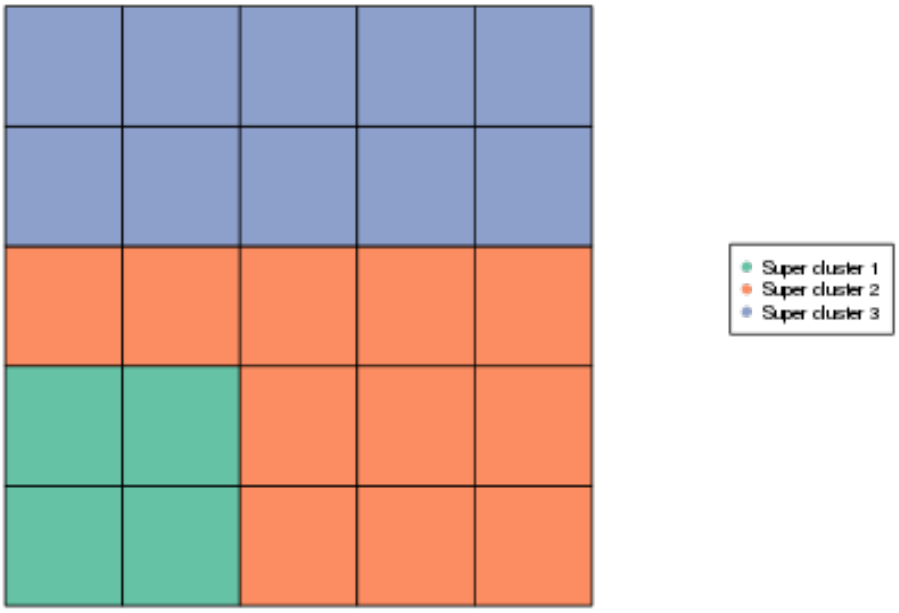


FIGURE 4.11 – Représentation des super classes en couleurs

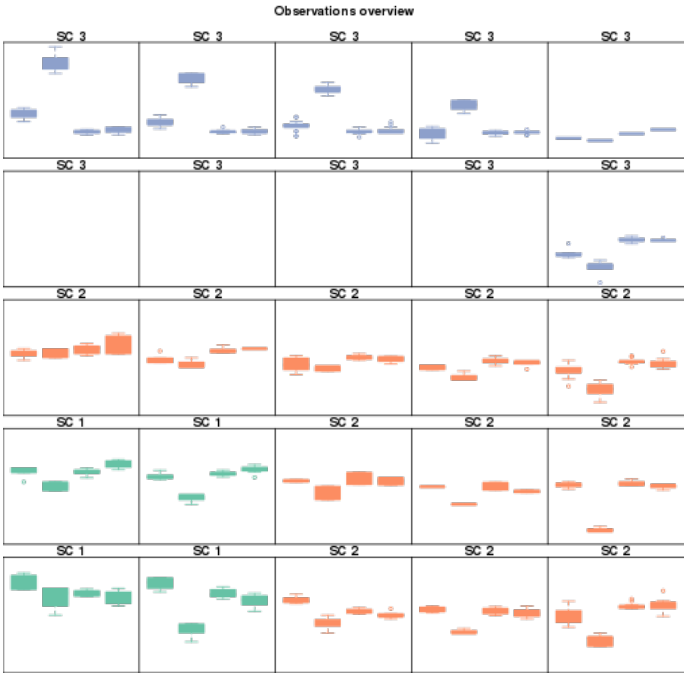


FIGURE 4.12 – Valeurs des observations identifiées par super classes

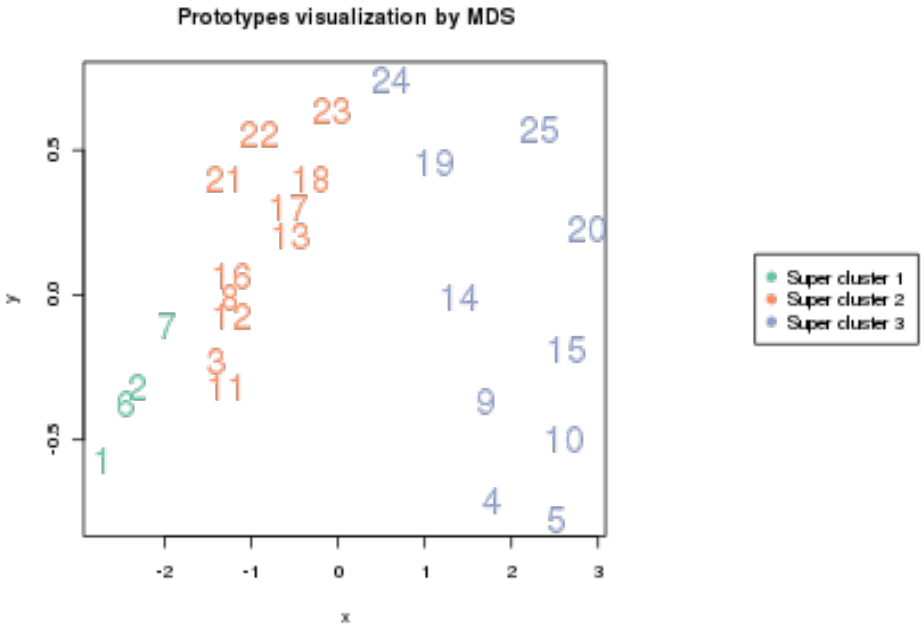


FIGURE 4.13 – Visualisation des prototypes projetés coloriés en fonction de la super classe

4.3.2 Illustration korresp : les données *presidentielles2002*

Le jeu de données `presidentielles2002`[6] contient le nombre de votes pour chaque département (lignes) et chaque candidat (colonnes) au premier tour de l'élection présidentielle de 2002.

Puisque ces données sont présentées sous forme de matrice de contingence, l'algorithme de cartes auto-organisatrices à appliquer n'est plus `numeric` mais `korresp` :

```
presi.som <- trainSOM(x.data = presidentielles2002, dimension = c(8, 8), type =
"korresp", scaling = "chi2")
```

L'ensemble des graphiques disponibles dans le cas `korresp` sont résumés par la figure 4.2 ; celle-ci signale d'ailleurs que le graphique `names` possède un comportement différent dans le cas `korresp` : les noms des colonnes s'ajoutent aux noms des lignes, comme cela est montré par la figure 4.14. La classification semble ici, assez pertinente, notamment puisque :

- la candidate `TAUBIRA` voisine des départements d'outre mer où elle avait réalisé un gros score ;
- les candidats `BESANCENOT`, `LAGUILLER` et `GLUCKSTEIN`, candidats d'extrême gauche, sont classés dans le même neurone ;
- les candidats `LE PEN` et `CHIRAC` possèdent un grand nombre de départements à proximité, traduisant des scores élevés dans ces départements : ils seront les deux candidats à accéder au second tour.

[6] source : Ministère de l'Intérieur, http://www.interieur.gouv.fr/Elections/Les-resultats/Presidentielles.

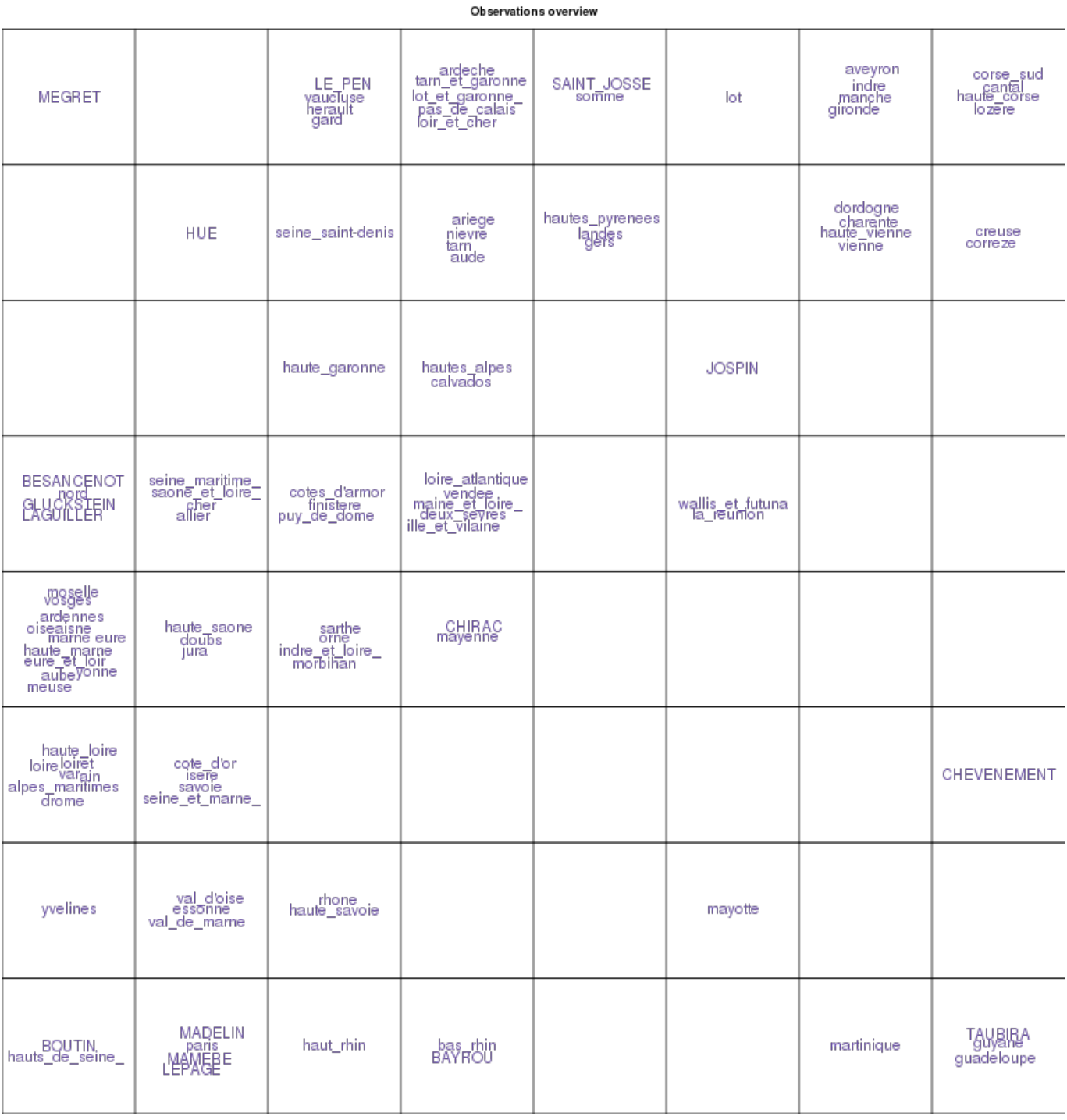


FIGURE 4.14 – Répartition des modalités

Le graphique `3d`, qui apporte la même imformation que le graphique `color`, est présenté couplé à ce dernier pour le candidat `CHIRAC` en figure 4.15.

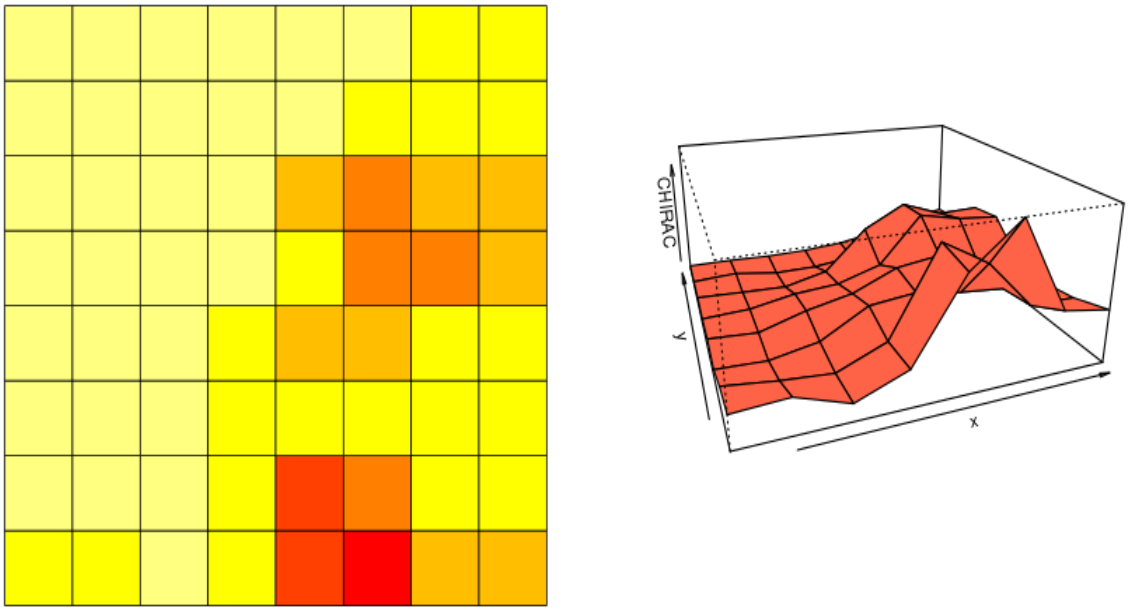


FIGURE 4.15 – Visualisation du nombre de votes pour CHIRAC

Le graphique `smooth.dist` (figure 4.16) représente, avec une palette de couleurs lissées, Les distances entre les prototypes de la carte. On remarque ici une forte cassure dans le coin bas droit de la carte, correspondante à la candidate `TAUBIRA` et aux départements d'outre mer. Les distances entre prototypes sont également visualisables avec une palette de couleurs, comme proposé par le graphique `umatrix` à partir de la moyenne des distances entre le prototype et ses voisins (voir figure 4.17).

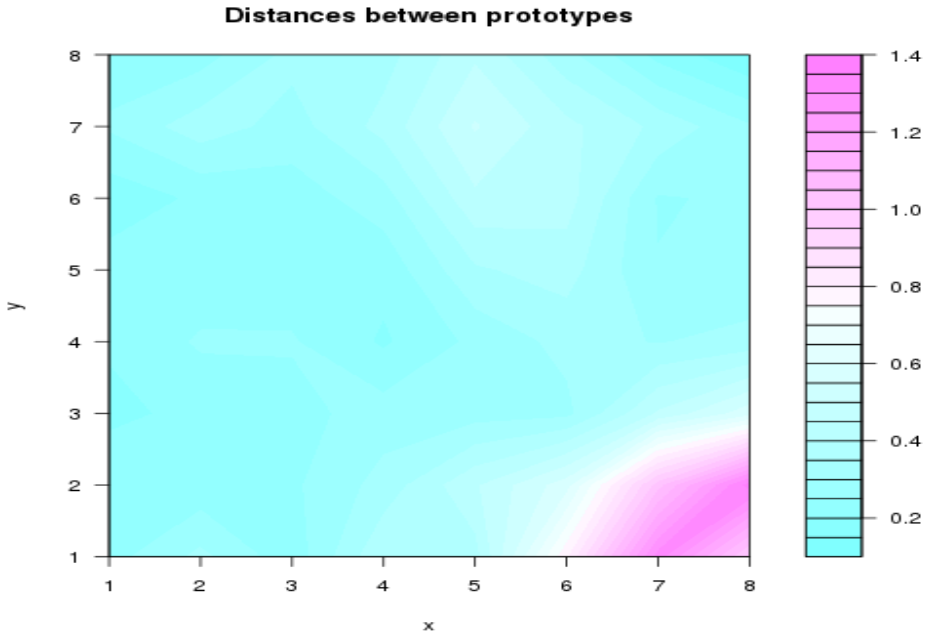


FIGURE 4.16 – Distances lissées entre prototypes

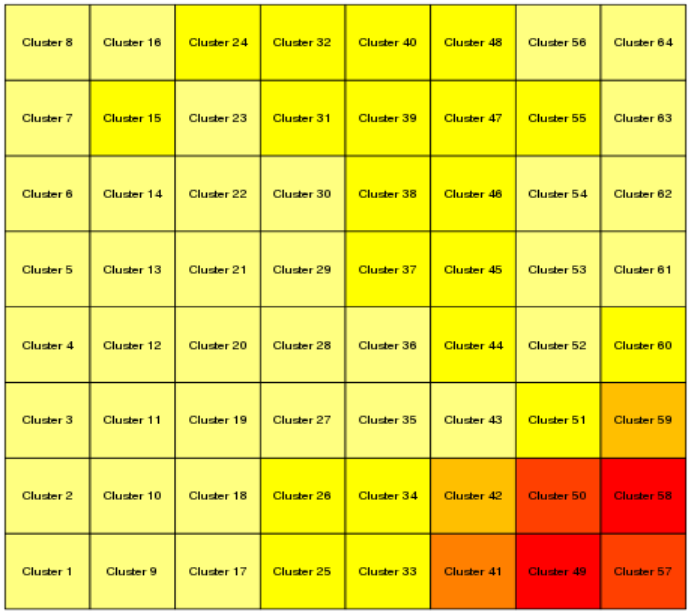


FIGURE 4.17 – Palette de couleur appliquée aux distances entre prototypes

Le cas `korresp`, comme le cas `numeric`, offre la possibilité d'effectuer une classification ascendante hiérarchique :

```
presi.sc <- superClass(presi.som, k=3)
```

Comme cela est précisé en section 4.1.2, il est possible dans le cas `korresp` de s'interesser soit aux variables lignes, soit aux variables colonnes. Les figures 4.18 et 4.19 couplent l'information de la super classification à la représentation des valeurs des prototypes, respectivement pour les départements et pour les candidats. Ils sont obtenus par appels au graphique `lines` comme suit :

```
plot(presi.sc, type="lines", print.title=TRUE, view="r") # "r" pour 'row'
plot(presi.sc, type="lines", print.title=TRUE, view="c") # "c" pour 'column'
```

Les 3 neurones appartenent à la super classe 3 possèdent des profils très différents, que ce soit pour les départements (départements d'outre mer) ou les candidats (`TAUBIRA`). Les super classes 1 et 2 ne semblent pas très différenciées pour les départements (figure 4.18) mais présentent en revanche des dissimilarités sur les profils des candidats (figure 4.19).

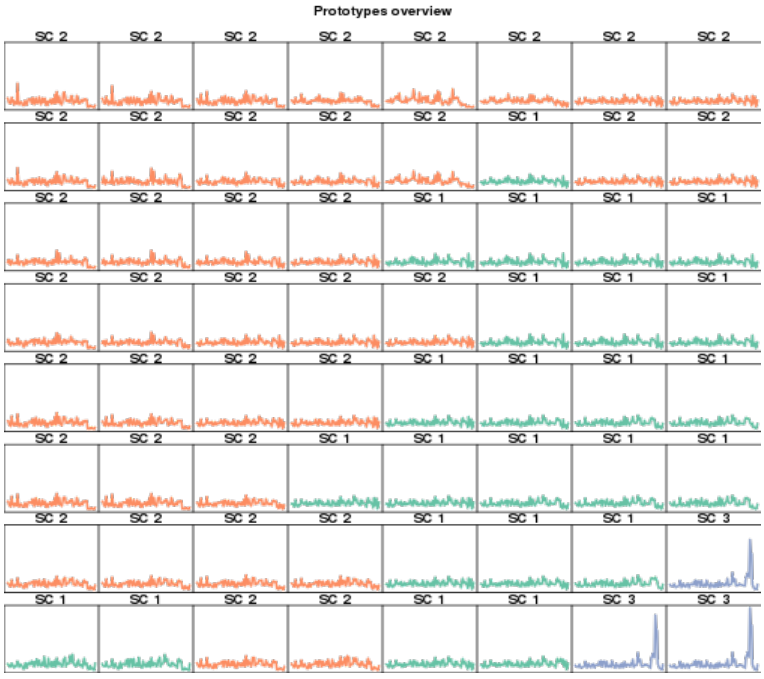


FIGURE 4.18 – Profils des départements avec super classification

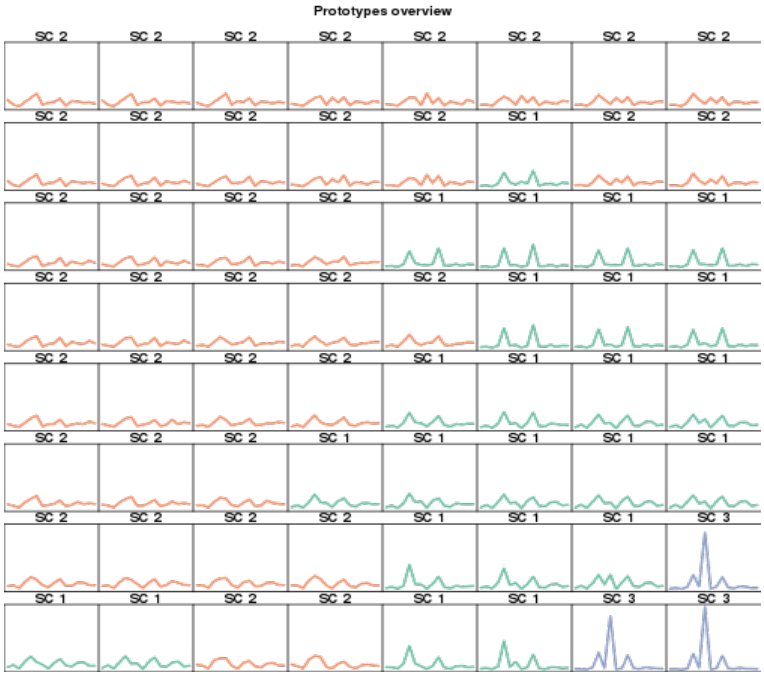


FIGURE 4.19 – Profils des candidats avec super classification

4.3.3 Illustration relationnelle : les données *lesmis*

Le jeu de données *lesmis*[7] est un graphe représentant les liens entre les personnages du roman Les Misérables de Victor Hugo. Un sommet correspond à un personnage, une arête signifie que les 2 personnages reliés apparaissent dans un même chapitre.

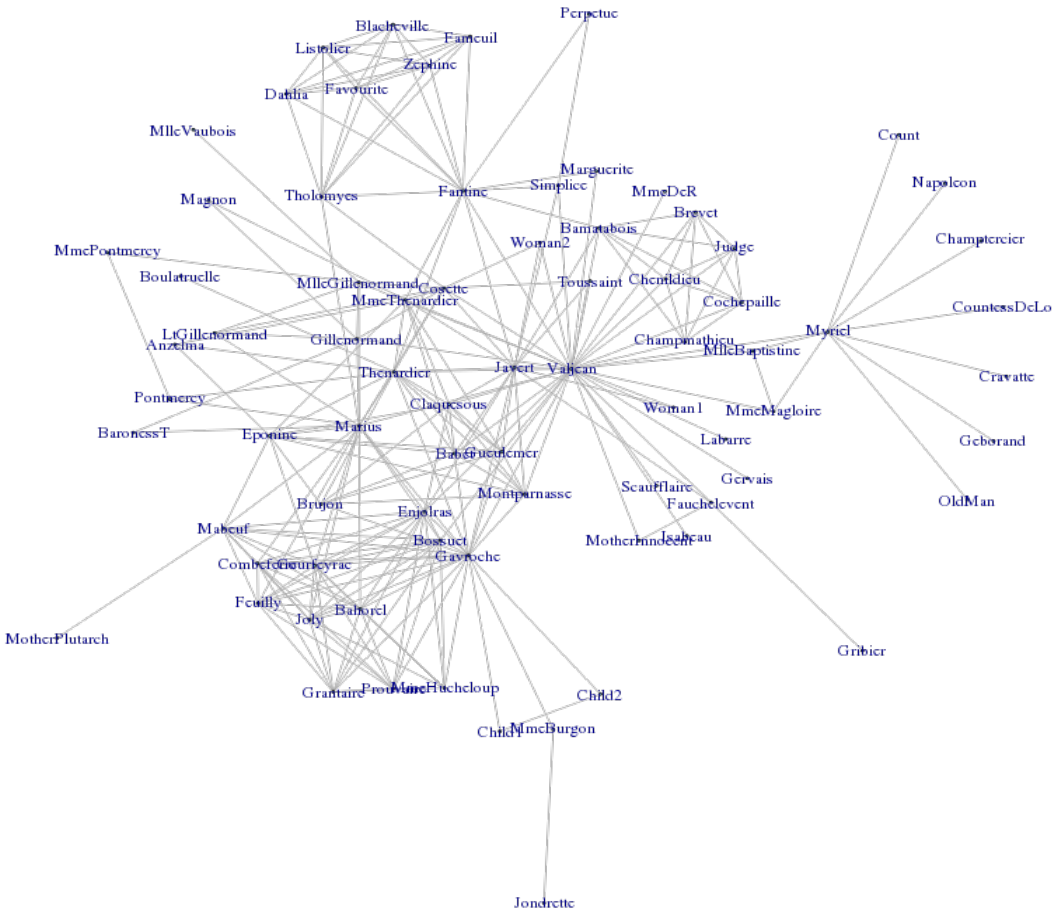


FIGURE 4.20 – Graphique de co apparition des personnages des Misérables

Pour pouvoir utiliser ces données pour le cas `'relational'` de l'algorithme, une matrice de dissimilarité a été calculée grâce à la fonction `shortest.path` du package **igraph**. À partir du graphe 4.20, une mesure de dissimilarité est définie, qui correspond au chemin le plus court entre les 2 personnages : cette dissimilarité est donc constituée de valeurs entières et positives.

```
mis.som <- trainSOM(x.data=dissim.lesmis, type="relational")
```

La distribution des personnages sur la carte est accessible soit avec un tableau, résultat de la fonction `table` de R :

[7] source : http://people.sc.fsu.edu/~jburkardt/datasets/sgb/jean.dat

```
table(mis.som$clustering)
##  1  2  3  4  5  6  7  9 11 12 14 15 18 20 21 22 24 25
##  6  7  3  2  8  1  3  3  3  7  3  3  2  6  6  5  3  6
```

Soit graphiquement, en appelant la fonction `plot.somRes` (voir figure 4.21) ou encore en utilisant une palette de couleur sur le graphe de départ (figure 4.22).

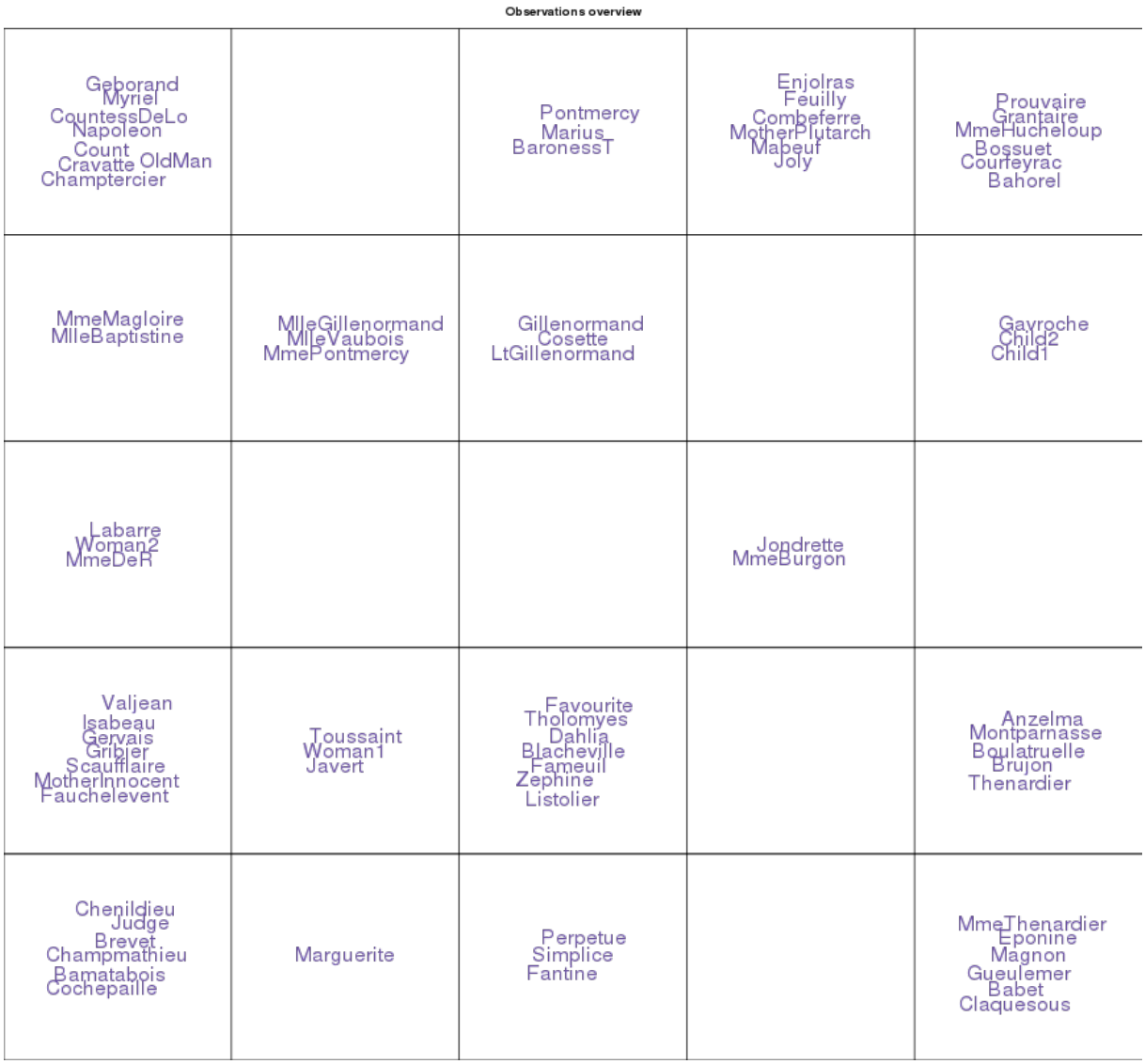


FIGURE 4.21 – Répartition des personnages sur la carte

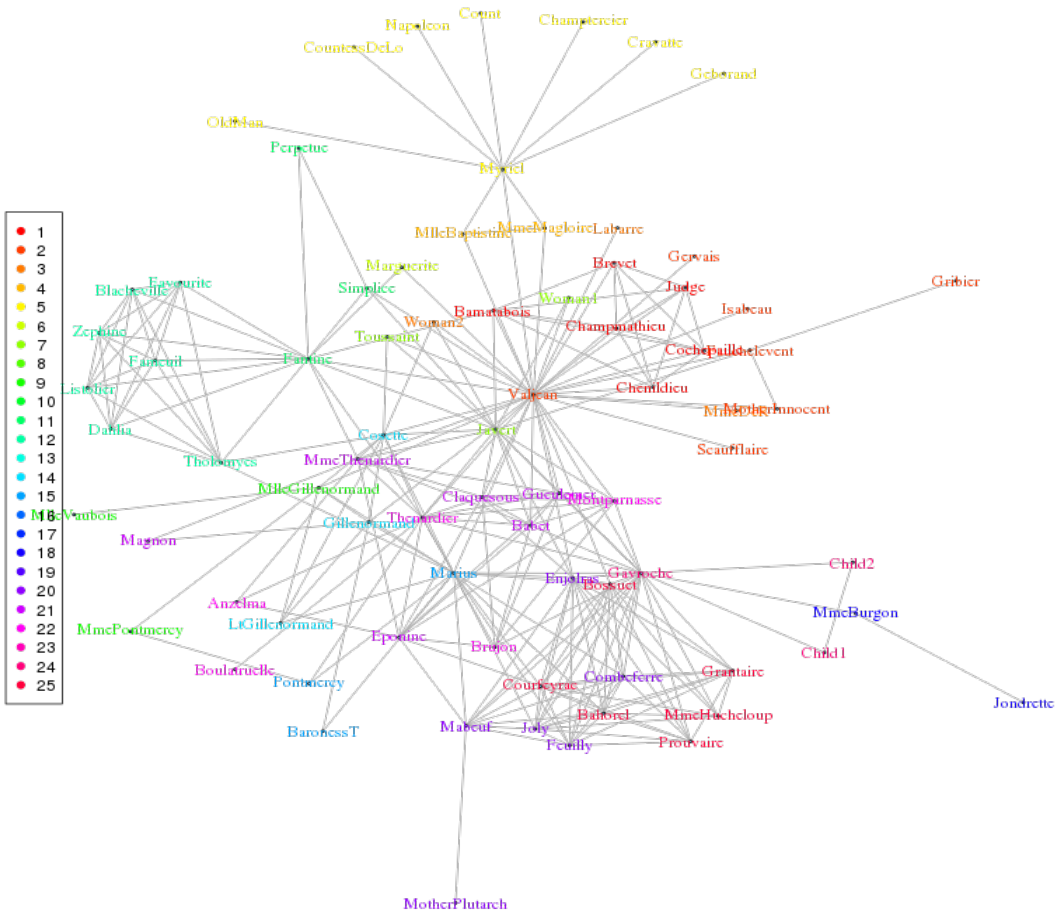


FIGURE 4.22 – Visualisation de la classification sur le graphe

La figure 4.23 représente le graphe projeté sur la grille : la taille de chaque sommet est proportionnelle au nombre de personnages classés dans le neurone et l'épaisseur de l'arête entre 2 neuronnes dépend du nombre d'arêtes entre personnages des 2 classes sur la figure 4.20.

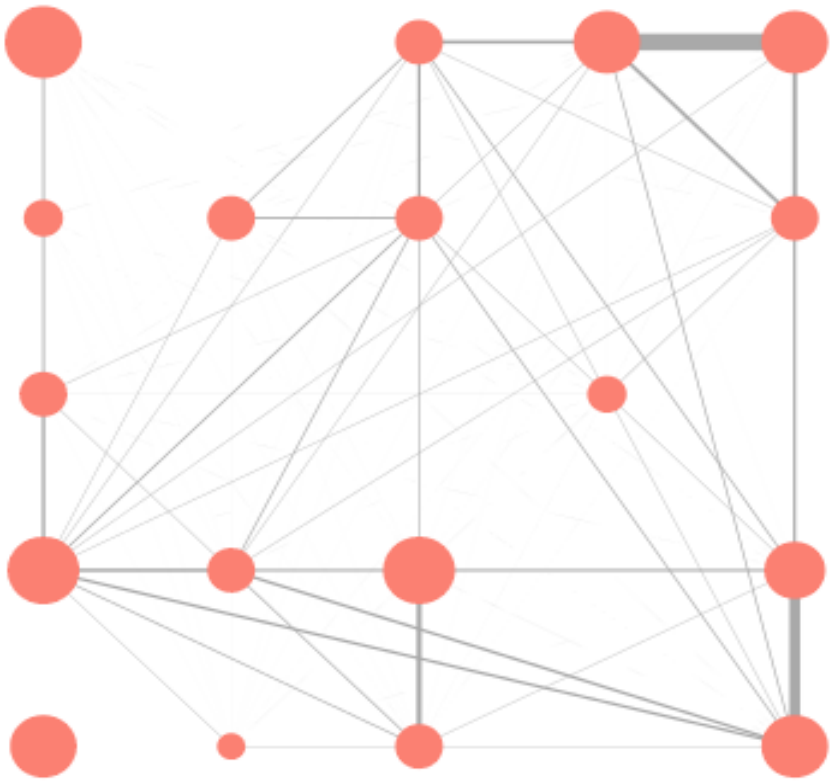


FIGURE 4.23 – Graphe projeté sur la grille

Les profils des observations peuvent, entre autres, être visualisés par un graphique de type `barplot`, présenté en figure 4.24.

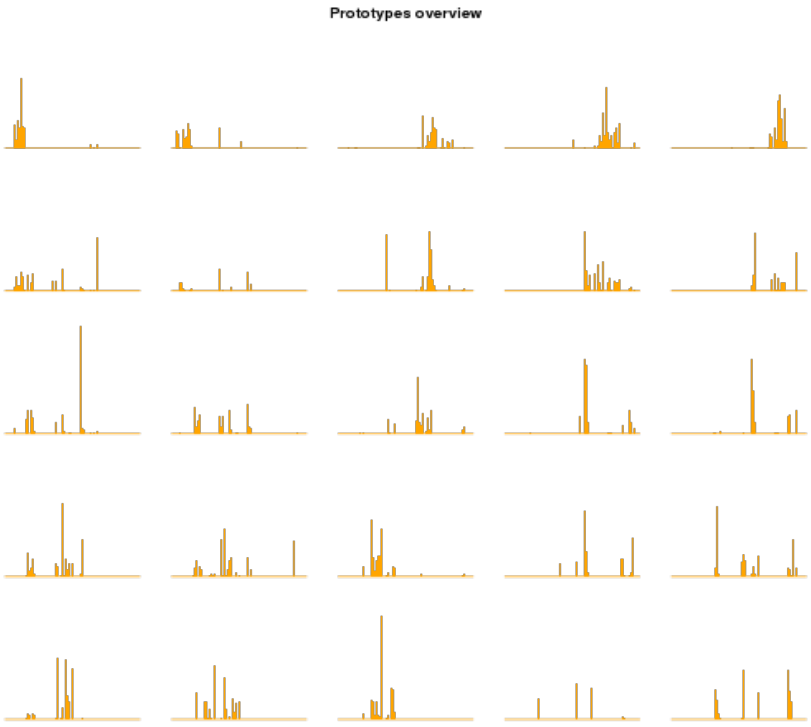


FIGURE 4.24 – Profils des personnages

Etant donné le faible nombre d'observations dans la majorité des classes, il devient pertinent de procéder à une super classification :

```
sc.mis <- superClass(mis.som, k=6)
```

Cela permet d'obtenir un nombre de classes réduit, qui produit donc des résultats plus proches de la réalité.

Outre le dendrogramme traditionnel produit par la graphique `dendrogram` (voir la figure iris-sc-dendro pour exemple), **SOMbrero** propose également une version en 3 dimensions de celui-ci via le graphique `dendro3d` (figure 4.25).

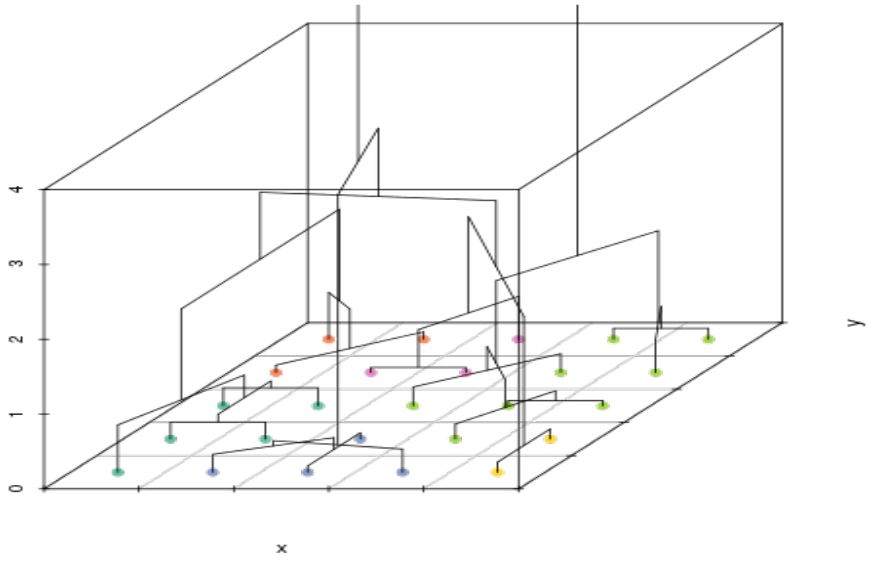


FIGURE 4.25 – Dendrogramme en 3 dimensions de la super classification

Ajouter l'information de la super classification au graphe de départ permet d'idenfier (figure 4.26) :

- deux des personnages principaux, Javert et Valjean, qui ont un rôle central dans le roman (super classe 1) ;
- les personnages qui interviennent uniquement dans l'histoire de l'évêque Myriel (super classe 2) ;
- Fantine et les personnages de son histoire (super classe 3) ;
- Marius et sa famille (Mme Pontmercy, sa mère, le Lieutenant Gillenormand, son père, etc) mais également Cosette, avec qui il aura une aventure (super classe 4) ;
- Gavroche, l'enfant abandonné des Thénardiers, et les personnages relatifs à son histoire (super classe 5) ;
- la famille Thénardier (super classe 6).

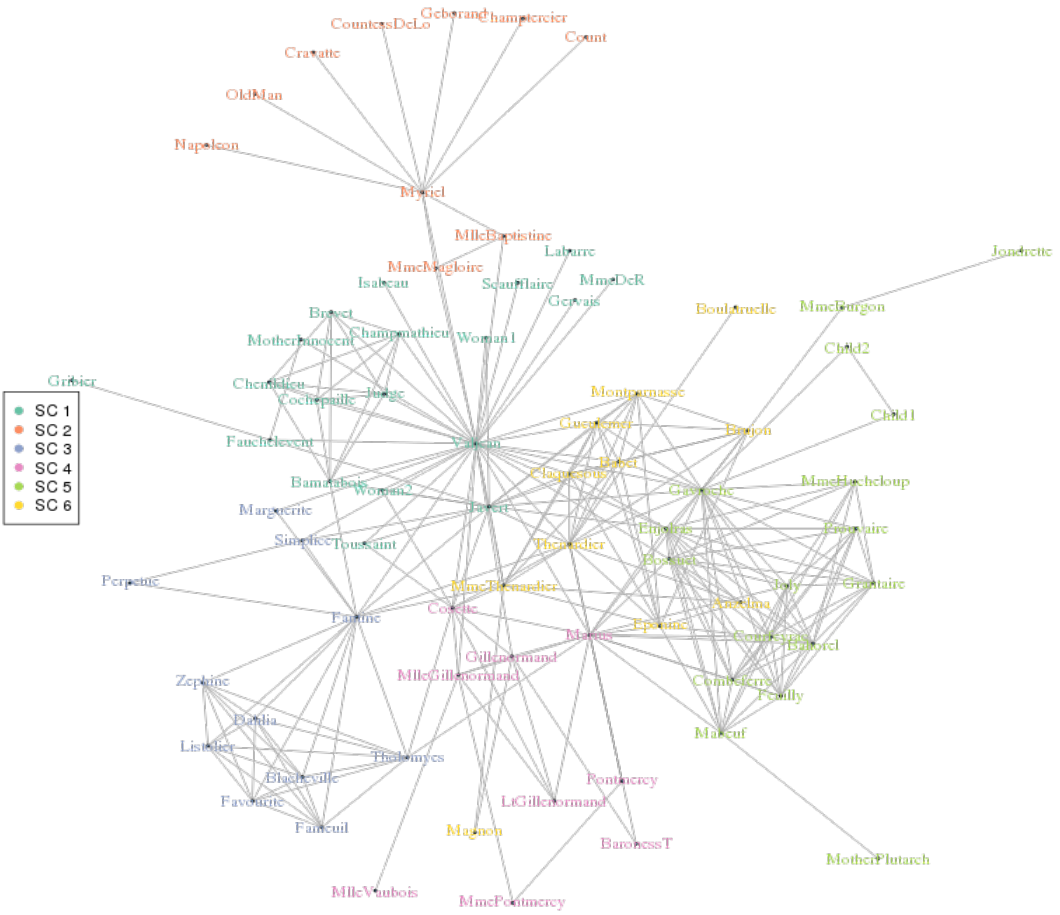


FIGURE 4.26 – Visualisation de la super classification sur le graphe

Chapitre 5

Conclusion

Ainsi, j'ai effectué mon stage de fin d'études Génie Informatique et Statistique pour le compte du laboratoire SAMM. Cette longue immersion dans le monde de l'entreprise m'a permis de mettre et pratique et d'approfondir mes connaissances théoriques, notamment en programmation et plus particulièrement en programmation R. Durant cette période, je me suis également confrontée aux réalités du monde du travail.

La réalisation du package **SOMbrero** m'a donné le loisir d'accomplir diverses tâches, principalement : acquisition de connaissances sur la méthode des cartes auto organisatrices, développement des scripts et documentations R, rédaction de guides utilisateurs et progrès sur le plan de la communication orale, dûs à de multiples présentations de mon travail en interne.

Cette expérience enrichissante et complète m'a offert une bonne préparation à mon insertion professionnelle et m'a rassurée quant à mes capacités dans le domaine informatique. En outre, j'ai pu aborder un aspect important du monde du travail actuel à savoir le travail à distance puisqu'une de mes encadrantes était basée à Paris.

Enfin, je tiens à souligner les bonnes conditions, matérielles et humaines, dans lesquelles s'est déroulé mon stage.

Bibliographie

[1] R Development Core Team. *R : A Language and Environment for Statistical Computing*. Vienna, Austria, 2012. ISBN 3-900051-07-0.

[2] L. Bendhaïba, M. Olteanu, and N. Villa-Vialaneix. SOMbrero : Cartes auto-organisatrices stochastiques pour l'intégration de données décrites par des tableaux de dissimilarités. In *2èmes Rencontres R*, pages 1–2, Lyon, France, 2013.

[3] T. Kohonen. *Self-Organizing Maps, 3rd Edition*, volume 30. Springer, Berlin, Heidelberg, New York, 2001.

[4] M. Cottrell, P. Letremy, and E. Roy. Analyzing a contingency table with Kohonen maps : a factorial correspondence analysis. In *Proceedings of IWANN'93, J. Cabestany, J. Mary, A. Prieto (Eds.), Lecture Notes in Computer Science*, pages 305–311. Springer Verlag, 1993.

[5] M. Olteanu, N. Villa-Vialaneix, and M. Cottrell. On-line relational som for dissimilarity data. In P.A. Estevez, J. Principe, P. Zegers, and G. Barreto, editors, *Advances in Self-Organizing Maps (Proceedings of WSOM 2012)*, volume 198 of *AISC (Advances in Intelligent Systems and Computing)*, pages 13–22, Santiago, Chile, 2012. Springer Verlag, Berlin, Heidelberg.

[6] C. Genolini. Construire un package. Classique ou S4. Technical report, 2009.

Annexe A

Scripts R

A.1 som.R

```
##### SOM algorithm functions
################################################################################

##### Auxiliary functions
################################################################################

calculateRadius <- function(the.grid, radius.type, ind.t, maxit) {
  ## TODO: implement other radius types
  # ind.t: iteration index
  if (radius.type=="letremy") {
    r0 <- max(c(floor(the.grid$dim[1]/2), floor(the.grid$dim[2]/2)))
    k <- 4*(r0-1)/maxit
    a <- floor(maxit/2)
    b <- floor(maxit*3/4)
    r <- ceiling(r0/(1+k*ind.t))
    if (ind.t==1) {
      r <- r0
    } else if (ind.t>=a & ind.t<b) {
      r <- 0.5
    } else if (ind.t>=b) {
      r <- 0
    }
  }
  r
}

selectNei <- function(the.neuron, the.grid, radius) {
  if (the.grid$dist.type=="letremy") {
    if (radius==0.5) {
      the.dist <- as.matrix(dist(the.grid$coord, diag=TRUE, upper=TRUE,
                                 method="euclidean"))[the.neuron,]
      the.nei <- which(the.dist<=1)
    } else {
      the.dist <- as.matrix(dist(the.grid$coord, diag=TRUE, upper=TRUE,
                                 method="maximum"))[the.neuron,]
      the.nei <- which(the.dist<=radius)
    }
  } else {
    the.dist <- as.matrix(dist(the.grid$coord, diag=TRUE, upper=TRUE,
                               method=the.grid$dist.type))[the.neuron,]
    the.nei <- which(the.dist<=radius)
  }
  the.nei
}

# Functions to manipulate objects in the input space
```

```
distEuclidean <- function(x,y) {
  sqrt(sum((x-y)^2))
}

distRelationalProto <- function(proto1, proto2, x.data) {
  -0.5*t(proto1-proto2)%*%x.data%*%(proto1-proto2)
}

calculateProtoDist <- function(prototypes, the.grid, type, complete=FALSE,
                               x.data=NULL) {
  if (!complete) {
    all.nei <- sapply(1:prod(the.grid$dim),selectNei,the.grid=the.grid,radius=1)
    all.nei <- sapply(1:prod(the.grid$dim), function(neuron)
      setdiff(all.nei[[neuron]],neuron))
    if (type!="relational") {# euclidean case
      distances <- sapply(1:prod(the.grid$dim), function(one.neuron) {
        apply(prototypes[all.nei[[one.neuron]],],1,distEuclidean,
              y=prototypes[one.neuron,])
      })
    } else {
      distances <- sapply(1:prod(the.grid$dim), function(one.neuron) {
        apply(prototypes[all.nei[[one.neuron]],],1,distRelationalProto,
              proto2=prototypes[one.neuron,], x.data=x.data)
      })
      if (sum(unlist(distances)<0)>0)
        warning("some of the relational 'distances' are negatives\n
  plots, qualities, super-clustering... may not work!",
                immediate.=TRUE, call.=TRUE)
    }
  } else {
    if (type=="relational") {# non euclidean case
      distances <- apply(prototypes,1,function(one.proto) {
        apply(prototypes, 1, distRelationalProto, proto2=one.proto,
              x.data=x.data)
      })
      if (sum(distances<0)>0)
        warning("some of the relational 'distances' are negatives\n
  plots, qualities, super-clustering... may not work!",
                immediate.=TRUE, call.=TRUE)
    } else distances <- as.matrix(dist(prototypes, upper=TRUE, diag=TRUE))
  }

  distances
}

## Functions used during training of SOM
# Step 2: Preprocess data ("korresp" case)
korrespPreprocess <- function(cont.table) {
  both.profiles <- matrix(0, nrow=nrow(cont.table)+ncol(cont.table),
                          ncol=ncol(cont.table)+nrow(cont.table))
  # row profiles
  both.profiles[1:nrow(cont.table), 1:ncol(cont.table)] <-
    cont.table/outer(rowSums(cont.table),
                     sqrt(colSums(cont.table)/sum(cont.table)))
  # column profiles
  both.profiles[(nrow(cont.table)+1):(nrow(cont.table)+ncol(cont.table)),
                (ncol(cont.table)+1):(ncol(cont.table)+nrow(cont.table))] <-
    t(cont.table)/outer(colSums(cont.table),
                        sqrt(rowSums(cont.table)/sum(cont.table)))
  # Best column to complete row profiles
  best.col <- apply(both.profiles[1:nrow(cont.table), 1:ncol(cont.table)],
                    1,which.max)
  both.profiles[1:nrow(cont.table), (ncol(cont.table)+1):ncol(both.profiles)] <-
    both.profiles[best.col+nrow(cont.table),
                  (ncol(cont.table)+1):ncol(both.profiles)]
  # Best row to complete col profiles
  best.row <- apply(both.profiles[(nrow(cont.table)+1):
                                    (nrow(cont.table)+ncol(cont.table)),
```

```
                                    (ncol(cont.table)+1):
                                      (ncol(cont.table)+nrow(cont.table))],
                      1,which.max)
  both.profiles[(nrow(cont.table)+1):(nrow(cont.table)+ncol(cont.table)),
                1:ncol(cont.table)] <-
    both.profiles[best.row, 1:ncol(cont.table)]
  # Names
  rownames(both.profiles) <- c(rownames(cont.table),colnames(cont.table))
  colnames(both.profiles) <- c(colnames(cont.table),rownames(cont.table))
  return(both.profiles)
}

# Step 3: Initialize prototypes
initProto <- function(parameters, norm.x.data, x.data) {
  if (is.null(parameters$proto0)) {
    if (parameters$init.proto=="random") {
      if (parameters$type=="relational") {
        prototypes <- t(apply(matrix(runif(prod(parameters$the.grid$dim,
                                           nrow(norm.x.data))),
                                    nrow=prod(parameters$the.grid$dim)),
                              1, function(x)x/sum(x)))
      } else {
        # both numeric and korresp
        prototypes <- sapply(1:ncol(norm.x.data),
                             function(ind){
                               runif(prod(parameters$the.grid$dim),
                                     min=min(norm.x.data[,ind]),
                                     max=max(norm.x.data[,ind]))})
      }
    } else if (parameters$init.proto=="obs") {
      if (parameters$type=="korresp"|parameters$type=="numeric") {
        prototypes <- norm.x.data[sample(1:nrow(norm.x.data),
                                         prod(parameters$the.grid$dim),
                                         replace=TRUE),]
      } else if (parameters$type=="relational") {
        prototypes <- matrix(0, nrow=prod(parameters$the.grid$dim),
                             ncol=ncol(norm.x.data))
        prototypes[cbind(1:nrow(prototypes),
                         sample(1:ncol(prototypes),nrow(prototypes),
                                replace=TRUE))] <- 1
      }
    }
  } else {
    prototypes <- switch(parameters$scaling,
                         "unitvar"=scale(parameters$proto0,
                                         center=apply(x.data,2,mean),
                                         scale=apply(x.data,2,sd)),
                         "center"=scale(parameters$proto0,
                                        center=apply(x.data,2,mean),
                                        scale=FALSE),
                         "none"=as.matrix(parameters$proto0),
                         "chi2"=as.matrix(parameters$proto0))
  }
  return(prototypes)
}

# Step 5: Randomly choose an observation
selectObs <- function(ind.t, ddim, type) {
  if (type=="korresp") {
    if (ind.t%%2==0) {
      rand.ind <- sample(1:ddim[1],1)
    } else rand.ind <- sample((ddim[1]+1):(ddim[1]+ddim[2]),1)
  } else rand.ind <- sample(1:ddim[1],1)
  return(rand.ind)
}

# Step 6: Assignment step
oneObsAffectation <- function(x.new, prototypes, type, x.data=NULL) {
```

```
  if (type=="relational") {
    the.neuron <- which.min(prototypes%*%x.new-
                              0.5*diag(prototypes%*%x.data%*%
                                         t(prototypes)))
  } else the.neuron <- which.min(apply(prototypes, 1, distEuclidean, y=x.new))

  the.neuron
}

# Step 7: Update of prototypes
prototypeUpdate <- function(type, the.nei, epsilon, prototypes, rand.ind,
                            sel.obs) {
  if (type=="relational") {
    indic <- matrix(0,nrow=length(the.nei),ncol=ncol(prototypes))
    indic[,rand.ind] <- 1
    prototypes[the.nei,] <- (1-epsilon)*prototypes[the.nei,] + epsilon*indic
  } else {
    prototypes[the.nei,] <- (1-epsilon)*prototypes[the.nei,] +
      epsilon*outer(rep(1,length(the.nei)), sel.obs)
  }
}

# Step 8: calculate intermediate energy
# TODO: It would probably be better to implement a function 'distEltProto'
calculateClusterEnergy <- function(cluster, x.data, clustering, prototypes,
                                   parameters, radius) {
  if (parameters$type=="numeric" || parameters$type=="korresp") {
    if (parameters$radius.type=="letremy") {
      the.nei <- selectNei(cluster, parameters$the.grid, radius)
      if (sum(clustering%in%the.nei)>0) {
        return(sum((x.data[which(clustering%in%the.nei),]-
                      outer(rep(1,sum(clustering%in%the.nei)),
                                 prototypes[cluster,]))^2))
      }
    }
  } else if (parameters$type=="relational") {
    if (parameters$radius.type=="letremy") {
      the.nei <- selectNei(cluster, parameters$the.grid, radius)
      if (sum(clustering%in%the.nei)>0) {
        return(sum(prototypes%*%x.data[,which(clustering%in%the.nei)]-0.5*
                     diag(prototypes%*%x.data%*%t(prototypes))))
      }
    }
  }
}

calculateEnergy <- function(x.data, clustering, prototypes, parameters, ind.t) {
  if (parameters$type=="numeric" || parameters$type=="korresp") {
    if (parameters$radius.type=="letremy") {
      radius <- calculateRadius(parameters$the.grid, parameters$radius.type,
                                ind.t, parameters$maxit)
      return(sum(unlist(sapply(1:nrow(prototypes), calculateClusterEnergy,
                               x.data=x.data, clustering=clustering,
                               prototypes=prototypes, parameters=parameters,
                               radius=radius)))/
               nrow(x.data)/nrow(prototypes))
    }
  } else if (parameters$type=="relational") {
    if (parameters$radius.type=="letremy") {
      radius <- calculateRadius(parameters$the.grid, parameters$radius.type,
                                ind.t, parameters$maxit)
      return(sum(unlist(sapply(1:nrow(prototypes), calculateClusterEnergy,
                              x.data=x.data, clustering=clustering,
                              prototypes=prototypes, parameters=parameters,
                              radius=radius)))/
              nrow(x.data)/nrow(prototypes))
    }
  }
```

```
}

##### Main function
################################################################################
trainSOM <- function (x.data, ...) {
  param.args <- list(...)
  ## Step 1: Parameters handling
  if (!is.matrix(x.data)) x.data <- as.matrix(x.data, rownames.force=TRUE)

  # Default dimension: nb.obs/10 with minimum equal to 5 and maximum to 10
  if (is.null(param.args$dimension)) {
    if (!is.null(param.args$type) && param.args$type=="korresp")
      param.args$dimension <-
      c(max(5,min(10,ceiling(sqrt((nrow(x.data)+ncol(x.data))/10)))),
        max(5,min(10,ceiling(sqrt((nrow(x.data)+ncol(x.data))/10)))))
    else
      param.args$dimension <- c(max(5,min(10,ceiling(sqrt(nrow(x.data)/10)))),
                  max(5,min(10,ceiling(sqrt(nrow(x.data)/10)))))
  }
  # Default maxit: nb.obs*5
  if (is.null(param.args$maxit)) {
    if (!is.null(param.args$type) && param.args$type=="korresp")
      param.args$maxit <- round((nrow(x.data)+ncol(x.data))*5)
    else
      param.args$maxit <- round(nrow(x.data)*5)
  }
  # Check inputs
  if (!is.null(param.args$type) && param.args$type=="relational" &&
        (!(identical(x.data, t(x.data))) || (sum(diag(x.data)!=0)>0)))
    stop("data do not match chosen SOM type ('relational')\n", call.=TRUE)

  # Initialize parameters and print
  parameters <- do.call("initSOM", param.args)
  if (parameters$verbose) {
    cat("Self-Organizing Map algorithm...\n")
    print.paramSOM(parameters)
  }

  # Check proto0 also now that the parameters have been initialized
  if (!is.null(param.args$proto0)) {
    if ((param.args$type=="korresp")&&
          (!identical(dim(param.args$proto0),
                      as.integer(c(prod(param.args$dimension),
                                   ncol(x.data)+nrow(x.data)))))) {
      stop("initial prototypes dimensions do not match SOM parameters:
         in the current SOM, prototypes must have ",
           prod(param.args$dimension), " rows and ",
           ncol(x.data)+nrow(x.data), " columns\n", call.=TRUE)
    } else if (!identical(dim(param.args$proto0),
                          as.integer(c(prod(param.args$dimension),
                                       ncol(x.data))))) {
      stop("initial prototypes dimensions do not match SOM parameters:
         in the current SOM, prototypes must have ",
           prod(param.args$dimension), " rows and ",
           ncol(x.data), " columns\n", call.=TRUE)
    }
  }

  ## Step 2: Preprocess the data
  # Scaling
  norm.x.data <- switch(parameters$scaling,
                        "unitvar"=scale(x.data, center=TRUE, scale=TRUE),
                        "center"=scale(x.data, center=TRUE, scale=FALSE),
                        "none"=as.matrix(x.data),
                        "chi2"=korrespPreprocess(x.data))

  ## Step 3: Initialize prototypes
  prototypes <- initProto(parameters, norm.x.data, x.data)
```

```
  # Step 4: Iitialize backup if needed
  if(parameters$nb.save>1) {
    backup <- list()
    backup$prototypes <- list()
    backup$clustering <- matrix(ncol=parameters$nb.save,
                                nrow=nrow(norm.x.data))
    backup$energy <- vector(length=parameters$nb.save)
    backup$steps <- round(seq(1,parameters$maxit,length=parameters$nb.save),0)
  }

  ## Main Loop: from 1 to parameters$maxit
  for (ind.t in 1:parameters$maxit) {
    if (parameters$verbose) {
      if (ind.t %in% round(seq(1, parameters$maxit, length=11))) {
        index <- match(ind.t, round(seq(1, parameters$maxit, length=11)))
        cat((index-1)*10, "% done\n")
      }
    }

    ## Step 5: Randomly choose an observation
    rand.ind <- selectObs(ind.t, dim(x.data), parameters$type)
    sel.obs <- norm.x.data[rand.ind,]

    ## Step 6: Assignment step
    # For the "korresp" type, cut the prototypes and selected observation
    if (parameters$type=="korresp") {
      if (ind.t%%2==0) {
        cur.obs <- sel.obs[1:ncol(x.data)]
        cur.prototypes <- prototypes[,1:ncol(x.data)]
      } else {
        cur.obs <- sel.obs[(ncol(x.data)+1):ncol(norm.x.data)]
        cur.prototypes <- prototypes[,(ncol(x.data)+1):ncol(norm.x.data)]
      }
    } else {
      cur.prototypes <- prototypes
      cur.obs <- sel.obs
    }
    # Assign
    winner <- oneObsAffectation(cur.obs, cur.prototypes, parameters$type,
                                norm.x.data)

    ## Step 7: Representation step
    # Radius value
    radius <- calculateRadius(parameters$the.grid, parameters$radius.type,
                              ind.t, parameters$maxit)
    the.nei <- selectNei(winner, parameters$the.grid, radius)
    # TODO: scale epsilon with a parameter???
    epsilon <- 0.3/(1+0.2*ind.t/prod(parameters$the.grid$dim))
    # Update
    prototypes[the.nei,] <- prototypeUpdate(parameters$type, the.nei, epsilon,
                                            prototypes, rand.ind, sel.obs)

    ## Step 8: Intermediate backups (if needed)
    if (parameters$nb.save==1) {
      warning("nb.save can not be 1\n No intermediate backups saved",
              immediate.=TRUE, call.=TRUE)
    }
    if (parameters$nb.save>1) {
      if(ind.t %in% backup$steps) {
        out.proto <- switch(parameters$scaling,
                            "unitvar"=scale(prototypes,
                                            center=-apply(x.data,2,mean)/
                                              apply(x.data,2,sd),
                                            scale=1/apply(x.data,2,sd)),
                            "center"=scale(prototypes,
                                           center=-apply(x.data,2,mean),
                                           scale=FALSE),
```

```
                            "none"=prototypes,
                            "chi2"=prototypes)
        colnames(out.proto) <- colnames(norm.x.data)
        rownames(out.proto) <- 1:prod(parameters$the.grid$dim)
        res <- list("parameters"=parameters, "prototypes"=out.proto,
                    "data"=x.data)
        class(res) <- "somRes"

        ind.s <- match(ind.t,backup$steps)
        backup$prototypes[[ind.s]] <- out.proto
        backup$clustering[,ind.s] <- predict.somRes(res, x.data)
        backup$energy[ind.s] <- calculateEnergy(norm.x.data,
                                                backup$clustering[,ind.s],
                                                prototypes, parameters, ind.t)
      }
      if (ind.t==parameters$maxit) {
        clustering <- backup$clustering[,ind.s]
        if (parameters$type=="korresp") {
          names(clustering) <- c(colnames(x.data), rownames(x.data))
        } else names(clustering) <- rownames(x.data)
        energy <- backup$energy[ind.s]
      }
    } else if (ind.t==parameters$maxit) {
      out.proto <- switch(parameters$scaling,
                          "unitvar"=scale(prototypes,
                                          center=-apply(x.data,2,mean)/
                                            apply(x.data,2,sd),
                                          scale=1/apply(x.data,2,sd)),
                          "center"=scale(prototypes,
                                         center=-apply(x.data,2,mean),
                                         scale=FALSE),
                          "none"=prototypes,
                          "chi2"=prototypes)

      res <- list("parameters"=parameters, "prototypes"=out.proto,
                  "data"=x.data)
      class(res) <- "somRes"
      clustering <- predict.somRes(res, x.data)
      if (parameters$type=="korresp") {
        names(clustering) <- c(colnames(x.data), rownames(x.data))
      } else names(clustering) <- rownames(x.data)
      energy <- calculateEnergy(norm.x.data, clustering, prototypes, parameters,
                                ind.t)
    }
  }

  colnames(out.proto) <- colnames(norm.x.data)
  rownames(out.proto) <- 1:prod(parameters$the.grid$dim)
  if (parameters$nb.save<=1) {
    res <- list("clustering"=clustering, "prototypes"=out.proto,
                "energy"=energy, "data"=x.data, "parameters"=parameters)
  } else {
    if (parameters$type=="korresp") {
      rownames(backup$clustering) <- c(colnames(x.data), rownames(x.data))
    } else rownames(backup$clustering) <- rownames(x.data)
    res <- list("clustering"=clustering, "prototypes"=out.proto,
                "energy"=energy, "backup"=backup, "data"=x.data,
                "parameters"=parameters)
  }
  class(res) <- "somRes"
  return(res)
}


## S3 methods for somRes class objects
##############################################################################

print.somRes <- function(x, ...) {
```

```
  cat("      Self-Organizing Map object...\n")
  cat("        ", x$parameters$mode, "learning, type:", x$parameters$type,"\n")
  cat("        ", x$parameters$the.grid$dim[1],"x",
      x$parameters$the.grid$dim[2],
      "grid with",x$parameters$the.grid$topo, "topology\n")
}

summary.somRes <- function(object, ...) {
  cat("\nSummary\n\n")
  cat("      Class : ", class(object),"\n\n")
  print(object)
  cat("\n      Final energy:", object$energy,"\n")
  if (object$parameters$type=="numeric") {
    cat("\n      ANOVA                 : \n")
    res.anova <- as.data.frame(t(sapply(1:ncol(object$data), function(ind) {
      c(round(summary(aov(object$data[,ind]~as.factor(object$clustering)))
              [[1]][1,4],digits=3),
        round(summary(aov(object$data[,ind]~as.factor(object$clustering)))
              [[1]][1,5],digits=8))
    })))
    names(res.anova) <- c("F", "pvalue")
    res.anova$significativity <- rep("",ncol(object$data))
    res.anova$significativity[res.anova$"pvalue"<0.05] <- "*"
    res.anova$significativity[res.anova$"pvalue"<0.01] <- "**"
    res.anova$significativity[res.anova$"pvalue"<0.001] <- "***"
    rownames(res.anova) <- colnames(object$data)

    cat("\n        Degrees of freedom : ",
        summary(aov(object$data[,1]~as.factor(object$clustering)))[[1]][1,1],
        "\n\n")
    print(res.anova)
    cat("\n")
  } else if (object$parameters$type=="korresp") {
    chisq.res <- chisq.test(object$data)
    if (chisq.res$p.value<0.05) sig <- "*"
    if (chisq.res$p.value<0.01) sig <- "**"
    if (chisq.res$p.value<0.001) sig <- "***"
    cat("\n     ", chisq.res$method, ":\n\n")
    cat("         X-squared               : ", chisq.res$statistic, "\n")
    cat("         Degrees of freedom      : ", chisq.res$parameter, "\n")
    cat("         p-value                 : ", chisq.res$p.value, "\n")
    cat("                  significativity : ", sig, "\n")
  }
}

predict.somRes <- function(object, x.new, ...) {
  if (is.null(dim(x.new))) x.new <- matrix(x.new,nrow=1,
                                           dimnames=list(1,
                                                         colnames(object$data)))
  if(object$parameters$type!="korresp") {
    norm.x.new <- switch(object$parameters$scaling,
                         "unitvar"=scale(x.new,
                                         center=apply(object$data,2,mean),
                                         scale=apply(object$data,2,sd)),
                         "center"=scale(x.new,
                                        center=apply(object$data,2,mean),
                                        scale=FALSE),
                         "none"=as.matrix(x.new))
    norm.proto <- switch(object$parameters$scaling,
                         "unitvar"=scale(object$prototypes,
                                         center=apply(object$data,2,mean),
                                         scale=apply(object$data,2,sd)),
                         "center"=scale(object$prototypes,
                                        center=apply(object$data,2,mean),
                                        scale=FALSE),
                         "none"=object$prototypes)
    winners <- apply(norm.x.new, 1, oneObsAffectation,
                     prototypes=norm.proto, type=object$parameters$type,
```

```
                    x.data=object$data)
  } else {
    if (!identical(as.matrix(x.new), object$data))
      warning("For 'korresp' SOM, predict.somRes function can only be called on
              the original data set\n'object' replaced",
              call.=TRUE)
    norm.x.data <- korrespPreprocess(object$data)
    winners.rows <- apply(norm.x.data[1:nrow(object$data),1:ncol(object$data)],
                          1, oneObsAffectation,
                          prototypes=object$prototypes[,1:ncol(object$data)],
                          type=object$parameters$type)
    winners.cols <- apply(norm.x.data[(nrow(object$data)+1):ncol(norm.x.data),
                                      (ncol(object$data)+1):ncol(norm.x.data)],
                          1, oneObsAffectation,
                          prototypes=object$prototypes[,(ncol(object$data)+1):
                                                         ncol(norm.x.data)],
                          type=object$parameters$type)
    winners <- c(winners.cols,winners.rows)
  }
  winners
}

protoDist.somRes <- function(object, mode=c("complete","neighbors"), ...) {
  mode <- match.arg(mode)
  complete <- (mode=="complete")
  if (object$parameters$type=="relational") {
    x.data <- object$data
  } else x.data <- NULL

  distances <- calculateProtoDist(object$prototypes, object$parameters$the.grid,
                                  object$parameters$type, complete, x.data)

  return(distances)
}

protoDist <- function(object, mode,...) {
  UseMethod("protoDist")
}
```

A.2 plots.R

```
### subfunctions
orderIndexes <- function(the.grid, type) {
  if(type=="3d") tmp <- 1:the.grid$dim[1]
  else tmp <- the.grid$dim[1]:1
  match(paste(rep(1:the.grid$dim[2],the.grid$dim[1]),"-",
              rep(tmp,
                  rep(the.grid$dim[2],the.grid$dim[1])),
              sep=""),
        paste(the.grid$coord[,1],"-",the.grid$coord[,2],
              sep=""))
}

averageByCluster <- function(x,clustering,the.grid) {
  mean.var <- matrix(NA,nrow=prod(the.grid$dim),ncol=ncol(x))
  ne.neurons <- which(as.character(1:prod(the.grid$dim))%in%
                        names(table(clustering)))
  mean.var[ne.neurons,] <- matrix(unlist(by(x,clustering,colMeans)),
                                  byrow=TRUE,ncol=ncol(x))
  colnames(mean.var) <- colnames(x)
  return(mean.var)
}

words2Freq <- function(words, clustering, the.grid, type) {
  if (type=="names") {
```

```
    freq.words <- matrix(0, ncol=length(unique(words)), nrow=prod(the.grid$dim))
    all.tables <- by(words, clustering, table)
    freq.words[as.numeric(sort(unique(clustering))),] <- matrix(unlist(
      all.tables), ncol=length(unique(words)), byrow=TRUE)
    colnames(freq.words) <- names(all.tables[[1]])
  } else if (type=="words") {
    freq.words <- matrix(0, ncol=ncol(words), nrow=prod(the.grid$dim))
    freq.words[as.numeric(sort(unique(clustering))),] <- apply(words,2,
                                                               function(word) {
      by(word,clustering,sum)
    })
    colnames(freq.words) <- colnames(words)
  }
  return(freq.words)
}

projectGraph <- function(the.graph, clustering, the.grid) {
  # TODO: improve it to handle weighted graphs and directed graphs
  all.neurons <- 1:prod(the.grid$dim)
  nonempty.neurons <- sort(unique(clustering))
  p.edges <- NULL
  p.edges.weights <- NULL
  v.sizes <- length(as.vector(V(the.graph)[which(clustering==
                                                   nonempty.neurons[1])]))
  for (neuron in nonempty.neurons[-1]) {
    v.neuron <- as.vector(V(the.graph)[which(clustering==neuron)])
    v.sizes <- c(v.sizes,length(v.neuron))
    for (neuron2 in setdiff(nonempty.neurons,1:neuron)) {
      p.edges <- c(p.edges, neuron, neuron2)
      v.neuron2 <- as.vector(V(the.graph)[which(clustering==neuron2)])
      p.edges.weights <- c(p.edges.weights,
                           length(E(the.graph)[from(v.neuron) & to(v.neuron2)]))
    }
  }
  proj.graph <- graph.data.frame(matrix(p.edges, ncol=2, byrow=TRUE),
                                 directed=FALSE,
                                 vertices=data.frame("name"=nonempty.neurons,
                                                     "sizes"=v.sizes))
  E(proj.graph)$nb.edges <- p.edges.weights
  proj.graph <- set.graph.attribute(proj.graph, "layout",
                                    the.grid$coord[nonempty.neurons,])
  return(proj.graph)
}

paramGraph <- function(the.grid, print.title, type) {
  if (print.title) {
    if(type%in%c("lines","pie","boxplot","names","words")) {
      the.mar <- c(0,0,1,0)
    } else the.mar <- c(2,1,1,1)
  } else {
    if(type%in%c("lines","pie","boxplot","names","words")) {
      the.mar <- c(0,0,0,0)
    } else the.mar <- c(2,1,0.5,1)
  }
  list("mfrow"=c(the.grid$dim[1], the.grid$dim[2]),"oma"=c(0,0,3,0),
       "bty"="c", "mar"=the.mar)
}

myTitle <- function(args, what) {
  if (is.null(args$main)) {
    the.title <- switch(what,
                        "prototypes"="Prototypes overview",
                        "obs"="Observations overview",
                        "add"="Additional variable overview")
  } else the.title <- args$main
  return(the.title)
}
```

```
plotOneVariable <- function(ind, var.val, type, the.title, args) {
  args$ylab <- ""
  args$xlab <- ""
  if (type!="names" && type!="words" && !is.null(args$col)
      && length(args$col)>1) {
    args$col <- args$col[ind]
    if ((type=="boxplot")&(!is.null(args$border)))
      args$border <- args$border[ind]
  }
  if (!is.null(the.title)) {
    args$main <- the.title
  } else args$main <- NULL
  if (all(is.na(var.val))) {
    plot(1,type="n",axes=F,xlab="",ylab="",main=args$main)
    if (type %in% c("lines","boxplot", "names", "words")) box()
  } else {
    if(type=="lines") {
      if (is.null(args$lwd)) args$lwd <- 2
      if (is.null(args$type)) args$type <- "l"
      if (is.null(args$col)) args$col <- "tomato"
      args$xaxt <- "n"
      args$yaxt <- "n"
      args$x <- var.val
      do.call("plot",args)
    } else if (type=="barplot") {
      args$height <- var.val
      args$axes <- FALSE
      if (is.null(args$col)) args$col <- "orange"
      if (is.null(args$names.arg)) args$names.arg <- ""
      if (is.null(args$border)) args$border <- FALSE
      do.call("barplot",args)
      abline(h=0, col=args$col)
    } else if (type=="boxplot") {
      args$x <- var.val
      args$axes <- FALSE
      if (is.null(args$names)) args$names <- FALSE
      if (is.null(args$col)) args$col <- "tan2"
      do.call("boxplot", args)
      box()
    } else if (type %in% c("names", "words")) {
      if (sum(var.val)>0) {
        if (is.null(args$min.freq)) args$min.freq <- 1
        args$words <- names(var.val)[var.val>0]
        args$freq <- sqrt(var.val[var.val>0])
        args$scale <- args$scale*max(var.val)
        if (is.null(args$ordered.colors)) args$ordered.colors <- TRUE
        if (is.null(args$rot.per)) args$rot.per <- 0
        do.call("wordcloud", args)
      } else plot(1,type="n",axes=F,xlab="",ylab="")
      box()
      title(the.title)
    }
  }
}

plotAllVariables <- function(what, type, values, clustering=NULL, print.title,
                             the.titles, is.scaled, the.grid, args) {
  par(paramGraph(the.grid, print.title, type))
  ordered.index <- orderIndexes(the.grid, type)
  if (!is.null(args$col) && length(args$col)>1 &&
        length(args$col)!=length(ordered.index)) {
    warning("unadequate number of colors; first color will be used for all\n",
            call.=TRUE, immediate.=TRUE)
    args$col <- args$col[1]
  }
  if (print.title) {
    the.titles <- the.titles
  } else the.titles <- rep(NULL,length(ordered.index))
```

```
  if (type %in% c("names", "words")) {
    freq.words <- words2Freq(values, clustering, the.grid, type)
    if (is.null(args$colors)) {
      nb.breaks <- 6
      all.colors <- brewer.pal(9,"Purples")[5:9]
    } else if (length(args$colors)==1) {
      words.col <- matrix(args$colors, ncol=ncol(freq.words),
                          nrow=nrow(freq.words))
    } else {
      nb.breaks <- length(args$colors)+1
      all.colors <- args$colors
    }
    if (is.null(args$colors)|length(args$colors)>1) {
      the.breaks <- seq(min(freq.words[freq.words>0])-0.1,
                        max(freq.words)+0.1,length=nb.breaks)
      words.cut <- apply(freq.words,2,cut,breaks=the.breaks,labels=FALSE)
      words.col <- apply(words.cut, 2, function(wc) all.colors[wc])
    }
    if (is.null(args$scale)) {
      args$scale <- c(2,0.5)/max(freq.words)
    } else args$scale <- args$scale/max(freq.words)
    sapply(ordered.index, function(ind) {
      cur.args <- args
      if (sum(freq.words[ind,])>0) {
        cur.args$colors <- words.col[ind,freq.words[ind,]>0]
      }
      plotOneVariable(ind, freq.words[ind,], type, the.titles[ind], cur.args)
    })
  } else {
    is.scaled.values <- scale(values,is.scaled, is.scaled)
    args$ylim <- range(is.scaled.values)
    if (what=="prototypes"|type=="boxplot") {
      mean.val <- is.scaled.values
    } else mean.val <- averageByCluster(is.scaled.values, clustering, the.grid)
    if (type=="boxplot") {
      sapply(ordered.index, function(ind) {
        plotOneVariable(ind,matrix(as.matrix(mean.val)[which(clustering==ind),],
                                   ncol=ncol(as.matrix(values))), type,
                        the.titles[ind], args)
      })
    } else {
      sapply(ordered.index, function(ind) {
        plotOneVariable(ind,mean.val[ind,], type, the.titles[ind], args)
      })
    }
  }
  title(main=myTitle(args, what), outer=TRUE)
  par(mfrow=c(1,1), oma=c(0,0,0,0), mar=c(5, 4, 4, 2)+0.1)
}

plotColor <- function(what, values, clustering, the.grid, my.palette,
                      print.title, the.titles, args) {
  if (what!="prototypes") {
    x <- rep(NA, prod(the.grid$dim))
    ne.neurons <- which(as.character(1:prod(the.grid$dim))%in%
                          names(table(clustering)))
    x[ne.neurons] <- tapply(values, clustering, mean)
  } else x <- values
  if (is.null(my.palette)) {
    nb.breaks <- min(c(floor(1 + (3.3*log(prod(the.grid$dim),base=10))),
                       9))
  } else nb.breaks <- length(my.palette)
  the.breaks <- seq(min(x, na.rm=TRUE)-10^(-5),
                    max(x, na.rm=TRUE)+10^(-5),
                    length=nb.breaks+1)
  if (is.null(my.palette)) {
    my.colors <- heat.colors(nb.breaks)[nb.breaks:1]
  } else my.colors <- my.palette
```

```
  vect.color <- my.colors[cut(x, the.breaks, labels=FALSE)]
  if (what!="prototypes") vect.color[is.na(vect.color)] <- "white"
  plot.args <- c(list(x=the.grid, neuron.col=vect.color),
                 args)
  do.call("plot.myGrid",plot.args)
  if (print.title) {
    text(x=the.grid$coord[,1], y=the.grid$coord[,2],
         labels=the.titles, cex=0.7)
  }
}

plotRadar <- function(x,the.grid,what,print.title,the.titles,args) {
  args$main <- myTitle(args, what)
  if (print.title) {
      args$labels <- the.titles
  } else args$labels <- ""
  if (is.null(args$key.labels)) args$key.labels <- colnames(x)
  args$x <- x
  args$nrow <- the.grid$dim[1]
  args$ncol <- the.grid$dim[2]
  args$locations <- the.grid$coord
  if (is.null(args$draw.segments)) args$draw.segments <- TRUE
  if (is.null(args$len)) args$len <- 0.4
  do.call("stars", args)
}

plot3d <- function(x, the.grid, type, variable, args) {
  args$x <- 1:the.grid$dim[2]
  args$y <- 1:the.grid$dim[1]
  args$z <- t(matrix(data=x[orderIndexes(the.grid, "3d"),variable],
                ncol=the.grid$dim[2],
                nrow=the.grid$dim[1], byrow=TRUE))
  if (is.null(args$theta)) args$theta <- -20
  if (is.null(args$phi)) args$phi <- 20
  if (is.null(args$expand)) args$expand <- 0.4
  if (is.null(args$xlab)) args$xlab <- colnames(the.grid$coord)[1]
  if (is.null(args$ylab)) args$ylab <- colnames(the.grid$coord)[2]
  if (is.null(args$zlab)) args$zlab <- colnames(x)[variable]
  if (is.null(args$lwd)) args$lwd <- 1.5
  if (is.null(args$col)) args$col <- "tomato"
  do.call("persp", args)
}

plotOnePolygon <- function(ind, values, the.grid, col) {
  ## FIX IT think about a more general way to implement it
  cur.coords <- the.grid$coord[ind,]
  if (the.grid$topo=="square") {
    neighbors <- match(paste(rep((cur.coords[1]-1):(cur.coords[1]+1),c(3,3,3)),
                             rep((cur.coords[2]-1):(cur.coords[2]+1),3)),
                       paste(the.grid$coord[,1],the.grid$coord[,2]))
    neighbors <- neighbors[-5]
    poly.coord <- matrix(nrow=8, ncol=2)
    poly.coord[is.na(neighbors),] <- tcrossprod(rep(1,sum(is.na(neighbors))),
                                                cur.coords)
    active.indexes <- setdiff(1:8,which(is.na(neighbors)))
    poly.coord[active.indexes,] <- t(sapply(seq_along(active.indexes),
                                            function(ind2) {
      n.ind <- neighbors[active.indexes[ind2]]
      n.coords <- the.grid$coord[n.ind,]
      cur.coords + (n.coords-cur.coords)*values[ind2]
    }))
    poly.coord <- poly.coord[c(1,4,6,7,8,5,3,2),]
    polygon(poly.coord, col=col)
  } else {
    stop("Sorry: 'polygon' is still to be implemented for this dist.type or/and
          this topology.", call.=TRUE)
  }
}
```

```
plotPolygon <- function(values, clustering, the.grid, my.palette, args) {
  plot.args <- c(list(x=the.grid),args)
  do.call("plot.myGrid",plot.args)
  if (is.null(my.palette)) {
    nb.breaks <- min(c(floor(1 + (3.3*log(prod(the.grid$dim),
                                          base=10))),9))
  } else nb.breaks <- length(my.palette)
  the.breaks <- seq(0,max(table(clustering)),length=nb.breaks+1)
  if (is.null(my.palette)) {
    my.colors <- heat.colors(nb.breaks)[nb.breaks:1]
  } else my.colors <- my.palette
  freq.clust <- sapply(1:prod(the.grid$dim), function(ind) sum(clustering==ind))
  vect.color <- my.colors[cut(freq.clust, the.breaks, labels=FALSE)]
  vect.color[is.na(vect.color)] <- "white"
  invisible(sapply(1:prod(the.grid$dim), function(ind) {
    plotOnePolygon(ind,values[[ind]],the.grid,vect.color[ind])
  }))
}

### SOM algorithm graphics
plotPrototypes <- function(sommap, type, variable, my.palette, print.title,
                           the.titles, is.scaled, view, args) {
  ## types : 3d, lines, barplot, radar, color, smooth.dist, poly.dist, umatrix,
  # mds

  # default value for type="lines"
  if (!is.element(type,c("3d","lines","barplot","radar","color", "poly.dist",
                         "umatrix", "smooth.dist", "mds", "grid.dist"))) {
    warning("incorrect type replaced by 'lines'\n", call.=TRUE,
            immediate.=TRUE)
    type <- "lines"
  }
  # relational control
  if (sommap$parameters$type=="relational" && type %in% c("color", "3d"))
    stop("prototypes/", type, " plot is not available for 'relational'\n",
         call.=TRUE)

  if (type=="lines" || type=="barplot") {
    if (sommap$parameters$type=="korresp") {
      if (view=="r")
        tmp.proto <- sommap$prototypes[,(ncol(sommap$data)+1):
                                       ncol(sommap$prototypes)]
      else
        tmp.proto <- sommap$prototypes[,1:ncol(sommap$data)]
    } else
      tmp.proto <- sommap$prototypes
    plotAllVariables("prototypes",type,tmp.proto,
                     print.title=print.title,the.titles=the.titles,
                     is.scaled=is.scaled,
                     the.grid=sommap$parameters$the.grid,args=args)
  } else if (type=="radar") {
    if (sommap$parameters$type=="korresp") {
      if (view=="r")
        tmp.proto <- sommap$prototypes[,(ncol(sommap$data)+1):
                                       ncol(sommap$prototypes)]
      else
        tmp.proto <- sommap$prototypes[,1:ncol(sommap$data)]
    } else
      tmp.proto <- sommap$prototypes
    plotRadar(tmp.proto, sommap$parameters$the.grid, "prototypes",
              print.title, the.titles, args)
  } else if (type=="color") {
    if (length(variable)>1) {
      warning("length(variable)>1, only first element will be considered\n",
              call.=TRUE, immediate.=TRUE)
      variable <- variable[1]
    }
```

```
    if (sommap$parameters$type=="korresp" & (is.numeric(variable))) {
      if (view=="r")
        tmp.var <- variable+ncol(sommap$data)
      else tmp.var <- variable
    } else tmp.var <- variable
    plotColor("prototypes", sommap$prototypes[,tmp.var], sommap$clustering,
              sommap$parameters$the.grid, my.palette, print.title, the.titles,
              args)
  } else if (type=="3d") {
    if (length(variable)>1) {
      warning("length(variable)>1, only first element will be considered\n",
              call.=TRUE, immediate.=TRUE)
      variable <- variable[1]
    }
    if (sommap$parameters$type=="korresp" & (is.numeric(variable))) {
      if (view=="r")
        tmp.var <- variable+ncol(sommap$data)
      else
        tmp.var <- variable
    } else
      tmp.var <- variable
    plot3d(sommap$prototypes, sommap$parameters$the.grid, type, tmp.var, args)
  } else if (type=="poly.dist") {
    values <- calculateProtoDist(sommap$prototypes, sommap$parameters$the.grid,
                                 sommap$parameters$type, FALSE, sommap$data)
    if (sommap$parameters$type=="relational") {
      if (sum(unlist(values)<0)>0) {
        stop("Impossible to plot 'poly.dist'!", call.=TRUE)
      } else values <- lapply(values,sqrt)
    }

    maxi <- max(unlist(values))
    values <- lapply(values, function(x) 0.429*((maxi-x)/maxi+0.05))
    plotPolygon(values, sommap$clustering, sommap$parameters$the.grid,
                my.palette, args)
    if (print.title) {
      text(x=sommap$parameters$the.grid$coord[,1]-0.1,
           y=sommap$parameters$the.grid$coord[,2]+0.1,
           labels=the.titles, cex=0.7)
    }
  } else if (type=="umatrix" || type=="smooth.dist") {
    values <- calculateProtoDist(sommap$prototypes, sommap$parameters$the.grid,
                                 sommap$parameters$type, FALSE, sommap$data)
    if (sommap$parameters$type=="relational") {
      if (sum(unlist(values)<0)>0) {
        stop("Impossible to plot 'smooth.dist'!", call.=TRUE)
      } else values <- lapply(values,sqrt)
    }
    values <- unlist(lapply(values,mean))
    if (type=="umatrix") {
      plotColor("prototypes", values, sommap$clustering,
                sommap$parameters$the.grid, my.palette, print.title, the.titles,
                args)
    } else {
      args$x <- 1:sommap$parameters$the.grid$dim[2]
      args$y <- 1:sommap$parameters$the.grid$dim[1]
      args$z <- matrix(data=values, nrow=sommap$parameters$the.grid$dim[2],
                       ncol=sommap$parameters$the.grid$dim[1], byrow=TRUE)
      if (is.null(args$color.palette)) args$color.palette <- cm.colors
      if (is.null(args$main)) args$main <- "Distances between prototypes"
      if (is.null(args$xlab)) args$xlab <- "x"
      if (is.null(args$ylab)) args$ylab <- "y"
      do.call("filled.contour", args)
    }
  } else if (type=="mds") {
    if (sommap$parameters$type=="relational") {
      the.distances <- calculateProtoDist(sommap$prototypes,
                                          sommap$parameters$the.grid,
```

```
                                        "relational", TRUE, sommap$data)
      if (sum(the.distances<0)>0) {
        stop("Impossible to plot 'MDS'!", call.=TRUE)
      } else the.distances <- sqrt(the.distances)
    }
    else the.distances <- dist(sommap$prototypes)
    proj.coord <- cmdscale(the.distances, 2)
    args$x <- proj.coord[,1]
    args$y <- proj.coord[,2]
    if (is.null(args$pch)) args$type <- "n"
    if (is.null(args$xlab)) args$xlab <- "x"
    if (is.null(args$ylab)) args$ylab <- "y"
    if (is.null(args$main)) args$main <- "Prototypes visualization by MDS"
    if (is.null(args$labels)) {
      the.labels <- as.character(1:nrow(sommap$prototypes))
    } else {
      the.labels <- args$labels
      args$labels <- NULL
    }
    if (is.null(args$col)) args$col <- "black"
    if (is.null(args$cex)) args$cex <- 1
    do.call("plot", args)
    text(proj.coord,the.labels,cex=args$cex,col=args$col)
  } else if (type=="grid.dist") {
    if (sommap$parameters$type=="relational") {
      the.distances <- calculateProtoDist(sommap$prototypes,
                                        sommap$parameters$the.grid,
                                        "relational", TRUE, sommap$data)
     if (sum(the.distances<0)>0) {
      stop("Impossible to plot 'grid.dist'!", call.=TRUE)
     } else {
       the.distances <- sqrt(the.distances)
       the.distances <- the.distances[lower.tri(the.distances)]
     }
    } else the.distances <- dist(sommap$prototypes)
    args$x <- as.vector(the.distances)
    args$y <- as.vector(dist(sommap$parameters$the.grid$coord))
    if (is.null(args$pch)) args$pch <- '+'
    if (is.null(args$xlab)) args$xlab <- "prototype distances"
    if (is.null(args$ylab)) args$ylab <- "grid distances"
    do.call("plot", args)
  } else stop("Sorry: this type is still to be implemented.", call.=TRUE)
}

plotObs <- function(sommap, type, variable, my.palette, print.title, the.titles,
                    is.scaled, view, args) {
  ## types : hitmap, lines, names, color, barplot, boxplot, radar

  # default value is type="hitmap"
  if (!is.element(type,c("hitmap", "lines", "names", "color", "radar",
                         "barplot", "boxplot"))) {
    warning("incorrect type replaced by 'hitmap'\n", call.=TRUE,
            immediate.=TRUE)
    type <- "hitmap"
  }

  # korresp control
  if (sommap$parameters$type=="korresp" && !(type%in%c("hitmap", "names"))) {
    warning("korresp SOM: incorrect type replaced by 'hitmap'\n", call.=TRUE,
            immediate.=TRUE)
    type <- "hitmap"
  }
  # relational control
  if (sommap$parameters$type=="relational" && !(type%in%c("hitmap", "names"))) {
    warning("relational SOM: incorrect type replaced by 'hitmap'\n",
            call.=TRUE, immediate.=TRUE)
    type <- "hitmap"
  }
```

```
if (type=="lines" || type=="barplot") {
  plotAllVariables("obs", type, sommap$data, sommap$clustering,
                   print.title, the.titles, is.scaled,
                   sommap$parameters$the.grid, args)
} else if (type=="color") {
  if (length(variable)>1) {
    warning("length(variable)>1, only first element will be considered\n",
            call.=TRUE, immediate.=TRUE)
    variable <- variable[1]
  }
  plotColor("obs", sommap$data[,variable], sommap$clustering,
            sommap$parameters$the.grid, my.palette, print.title, the.titles,
            args)
} else if (type=="radar") {
  mean.var <- averageByCluster(sommap$data, sommap$clustering,
                               sommap$parameters$the.grid)
  plotRadar(mean.var, sommap$parameters$the.grid, "obs", print.title,
            the.titles, args)
} else if (type=="hitmap") {
  freq <- sapply(1:nrow(sommap$prototypes), function(ind) {
    length(which(sommap$clustering==ind))
  })
  freq <- freq/sum(freq)
  # basesize is 0.45 for the maximum frequence
  basesize <- 0.45*sqrt(freq)/max(sqrt(freq))

  if (is.null(args$col)) {
    my.colors <- rep("pink", nrow(sommap$prototypes))
  } else if (length(args$col)==1) {
    my.colors <- rep(args$col, nrow(sommap$prototypes))
  } else {
    if(length(args$col)==nrow(sommap$prototypes)){
      my.colors <- args$col
    } else {
      warning("unadequate number of colors default color will be used\n",
              immediate.=TRUE, call.=TRUE)
      my.colors <- rep("pink", nrow(sommap$prototypes))
    }
  }
  plot.args <- c(list(x=sommap$parameters$the.grid), args)
  do.call("plot.myGrid",plot.args)
  invisible(sapply(1:nrow(sommap$prototypes), function(ind){
    xleft <- (sommap$parameters$the.grid$coord[ind,1]-basesize[ind])
    xright <- (sommap$parameters$the.grid$coord[ind,1]+basesize[ind])
    ybottom <- (sommap$parameters$the.grid$coord[ind,2]-basesize[ind])
    ytop <- (sommap$parameters$the.grid$coord[ind,2]+basesize[ind])
    rect(xleft,ybottom,xright,ytop, col=my.colors[ind], border=NA)
  }))
} else if (type=="boxplot") {
  if (length(variable)>5) {
    stop("maximum number of variables for type='boxplot' exceeded\n",
         call.=TRUE)
  }
  plotAllVariables("obs", type, sommap$data[,variable], sommap$clustering,
                   print.title, the.titles, is.scaled,
                   sommap$parameters$the.grid, args)
} else if (type=="names") {
  if (sommap$parameters$type=="korresp") {
    values <- names(sommap$clustering)
  } else {
    if (!is.null(rownames(sommap$data))) {
      values <- rownames(sommap$data)
    } else values <- 1:nrow(sommap$data)
  }
  plotAllVariables("obs", type, values, sommap$clustering,
                   print.title, the.titles, is.scaled,
                   sommap$parameters$the.grid, args)
```

```
  }
}

plotEnergy <- function(sommap, args) {
  # possible only if some intermediate backups have been done
  if (is.null(sommap$backup)) {
    stop("no intermediate backups have been registered\n", call.=TRUE)
  } else {
    if (is.null(args$main))
      args$main <- "Energy evolution"
    if (is.null(args$ylab)) args$ylab <- "Energy"
    if (is.null(args$xlab)) args$xlab <- "Steps"
    if (is.null(args$type)) args$type <- "b"
    if (is.null(args$pch)) args$pch <- "+"
    args$x <- sommap$backup$steps
    args$y <- sommap$backup$energy
    do.call("plot",args)
  }
}

plotAdd <- function(sommap, type, variable, proportional, my.palette,
                    print.title, the.titles, is.scaled, s.radius, pie.graph,
                    pie.variable, view, args) {
  ## types : pie, color, lines, boxplot, names, words, graph, barplot, radar
  # to be implemented: graph

  # default value is type="pie"
  if (!is.element(type,c("pie", "color", "lines", "barplot", "words",
                         "boxplot", "names", "radar", "graph"))) {
    warning("incorrect type replaced by 'pie'\n", call.=TRUE,
            immediate.=TRUE)
    type <- "pie"
  }
  if (is.null(variable)) {
    stop("for what='add', the argument 'variable' must be supplied\n",
         call.=TRUE)
  }

  # korresp control
  if (sommap$parameters$type=="korresp")
    stop("graphics of type 'add' do not exist for 'korresp'\n", call.=TRUE)

  if(type!="graph" && nrow(variable)!=nrow(sommap$data)){
    stop("length of additional variable does not fit length of the original
         data", call.=TRUE)
  }
  if (type=="pie") {
    if (!is.factor(variable)) variable <- as.factor(variable)
    cluster.freq <- tapply(variable,sommap$clustering,table)
    cluster.size <- table(sommap$clustering)
    par(paramGraph(sommap$parameters$the.grid, print.title, "pie"))
    ordered.index <- orderIndexes(sommap$parameters$the.grid, type)
    for (ind in ordered.index) {
      cur.cluster.vect.full <- cluster.freq[[as.character(ind)]]
      if (!is.null(cur.cluster.vect.full)) {
        cur.cluster.vect <- cur.cluster.vect.full[cur.cluster.vect.full>0]
        cur.args <- args
        cur.args$x <- cur.cluster.vect
        if (print.title) {
          cur.args$main <- the.titles[ind]
        } else cur.args$main <- NULL
        if (is.null(args$col)) {
          cur.args$col <- rainbow(nlevels(variable))[cur.cluster.vect.full>0]
        } else cur.args$col <- args$col[cur.cluster.vect.full>0]
        if (is.null(args$labels)) {
          cur.args$labels <- levels(variable)[cur.cluster.vect.full>0]
        } else cur.args$labels <- args$labels[cur.cluster.vect.full>0]
        if (proportional) {
```

```
          cur.args$radius=0.9*s.radius*sqrt(cluster.size[as.character(ind)]/
                                            max(cluster.size))
        } else {
          if (is.null(args$radius)) cur.args$radius <- 0.7
        }
        do.call("pie",cur.args)
      } else plot(1, type="n", bty="n", axes=FALSE)
    }
    if (is.null(args$main)) {
      title(main="Additional variable distribution", outer=TRUE)
    } else title(args$main, outer=TRUE)
    par(mfrow=c(1,1), oma=c(0,0,0,0), mar=c(5, 4, 4, 2)+0.1)
  } else if (type=="color") {
    if (!is.numeric(variable)) {
      stop("for type='color', argument 'variable' must be a numeric vector\n",
           call.=TRUE)
    }
    plotColor("add", variable, sommap$clustering, sommap$parameters$the.grid,
              my.palette, print.title, the.titles, args)
  } else if (type=="lines" || type=="barplot") {
    if (!all(apply(variable,2,is.numeric))) {
      stop("for type='lines' or 'barplot', argument 'variable' must be either a
           numeric matrix or a numeric data frame\n", call.=TRUE)
    }
    plotAllVariables("add", type, variable, sommap$clustering,
                     print.title, the.titles, is.scaled,
                     sommap$parameters$the.grid, args)
  } else if (type=="radar") {
    if (ncol(variable)<2) {
      stop("for type='radar', argument 'variable' must have at least 2
           columns\n")
    }
    mean.var <- averageByCluster(variable, sommap$clustering,
                                 sommap$parameters$the.grid)
    plotRadar(mean.var, sommap$parameters$the.grid, "add", print.title,
              the.titles, args)
  } else if (type=="boxplot") {
    if(ncol(variable)>5) {
      stop("maximum number of variables (5) for type='boxplot' exceeded\n",
           call.=TRUE)
    }
    plotAllVariables("add", type, variable, sommap$clustering,
                     print.title, the.titles, is.scaled,
                     sommap$parameters$the.grid, args)
  } else if (type=="words") {
    if (is.null(colnames(variable))) {
      stop("no colnames for 'variable'", call.=TRUE)
    }
    plotAllVariables("add", type, variable, sommap$clustering, print.title,
                     the.titles, is.scaled, sommap$parameters$the.grid, args)
  } else if (type=="names") {
    if (ncol(variable) != 1) {
      stop("for type='names', argument 'variable' must be a vector", call.=TRUE)
    }
    plotAllVariables("add", type, as.character(variable), sommap$clustering,
                     print.title, the.titles, is.scaled,
                     sommap$parameters$the.grid, args)
  } else if (type=="graph") {
    if (!is.igraph(variable)){
      stop("for type='graph', argument 'variable' must be an igraph object\n",
           call.=TRUE)
    }
    if (length(V(variable)) != nrow(sommap$data)){
      stop("length of additional variable does not fit length of the original
         data", call.=TRUE)
    }
    proj.graph <- projectGraph(variable, sommap$clustering,
                               sommap$parameters$the.grid)
```

```
    args$x <- proj.graph
    args$edge.width <- E(proj.graph)$nb.edges/max(E(proj.graph)$nb.edges)*10
    if (is.null(s.radius)) s.radius <- 1
    args$vertex.size <- s.radius*20*sqrt(V(proj.graph)$sizes)/
      max(sqrt(V(proj.graph)$sizes))
    if (is.null(args$vertex.label) & !print.title) args$vertex.label <- NA
    if (is.null(args$vertex.label) & print.title)
      args$vertex.label <- the.titles[as.numeric(V(proj.graph)$name)]
    if (pie.graph) {
      if (is.null(pie.variable)) {
        stop("pie.graph is TRUE, you must supply argument 'pie.variable'\n",
             call.=TRUE)
      }
      if (nrow(as.matrix(pie.variable)) != nrow(sommap$data)){
        stop("length of argument 'pie.variable' does not fit length of the
             original data", call.=TRUE)
      }
      args$vertex.shape <- "pie"
      if (!is.factor(pie.variable)) pie.variable <- as.factor(pie.variable)
      args$vertex.pie <- lapply(split(pie.variable, factor(sommap$clustering)),
                                table)
      if (is.null(args$vertex.pie.color)) {
        args$vertex.pie.color <- list(c(brewer.pal(8,"Set2"),
                                        brewer.pal(12,"Set3"))
                                        [1:nlevels(pie.variable)])
      }
    } else {
      if (is.null(args$vertex.color))
        args$vertex.color <- brewer.pal(12,"Set3")[4]
      if (is.null(args$vertex.frame.color))
        args$vertex.frame.color <- brewer.pal(12,"Set3")[4]
    }
    par(bg="white")
    do.call("plot.igraph", args)
  } else
    stop("Sorry: this type is still to be implemented.", call.=TRUE)
}

plot.somRes <- function(x, what=c("obs", "prototypes", "energy", "add"),
                        type=switch(what,
                                    "obs"="hitmap",
                                    "prototypes"="color",
                                    "add"="pie",
                                    "energy"=NULL),
                        variable = if (what=="add") NULL else
                          if (type=="boxplot") 1:min(5,ncol(x$data)) else 1,
                        my.palette=NULL,
                        is.scaled = if (x$parameters$type=="numeric") TRUE else
                          FALSE,
                        proportional=TRUE, print.title=FALSE, s.radius=1,
                        pie.graph=FALSE, pie.variable=NULL,
                        the.titles=if (what!="energy")
                          switch(type,
                                 "graph"=
                                   1:prod(x$parameters$the.grid$dim),
                                 paste("Cluster",
                                       1:prod(x$parameters$
                                              the.grid$dim))),
                        view = if (x$parameters$type=="korresp") "r" else NULL,
                        ...) {
  args <- list(...)
  what <- match.arg(what)
  if ((x$parameters$type=="korresp")&&!(view%in%c("r","c")))
      stop("view must be one of 'r'/'c'",call.=TRUE)
  if (length(the.titles)!=prod(x$parameters$the.grid$dim) & what!="energy") {
    the.titles=switch(type,
                      "graph"=1:prod(x$parameters$the.grid$dim),
                      paste("Cluster",1:prod(x$parameters$the.grid$dim)))
```

```
    warning("unadequate length for 'the.titles'; replaced by default",
            call.=TRUE, immediate.=TRUE)
  }

  switch(what,
         "prototypes"=plotPrototypes(x, type, variable, my.palette, print.title,
                                     the.titles, is.scaled, view, args),
         "energy"=plotEnergy(x, args),
         "add"=plotAdd(x, type, if (type!="graph") as.matrix(variable) else
           variable, proportional, my.palette, print.title, the.titles,
                       is.scaled, s.radius, pie.graph, pie.variable, view,
                       args),
         "obs"=plotObs(x, type, variable, my.palette, print.title, the.titles,
                       is.scaled, view, args))
}
```

A.3 quality.R

```
topographicError <- function (sommap) {
  if (sommap$parameters$type=="numeric") {
    all.dist <- apply(sommap$data, 1, function(x) {
      apply(sommap$prototypes,1,function(y) sum((x-y)^2) )
    })
    ind.winner2 <- apply(all.dist,2,function(x) order(x)[2])
  } else if (sommap$parameters$type=="korresp") {
    norm.data <- korrespPreprocess(sommap$data)
    nr <- nrow(sommap$data)
    nc <- ncol(sommap$data)
    all.dist.row <- apply(norm.data[1:nr,1:nc], 1, function(x) {
      apply(sommap$prototypes[,1:nc], 1, function(y) sum((x-y)^2) )
    })
    all.dist.col <- apply(norm.data[(nr+1):(nr+nc),(nc+1):(nr+nc)],
                          1, function(x) {
      apply(sommap$prototypes[,(nc+1):(nr+nc)], 1, function(y) sum((x-y)^2) )
    })
    ind.winner2 <- c(apply(all.dist.col,2,function(x) order(x)[2]),
                     apply(all.dist.row,2,function(x) order(x)[2]))
  } else if (sommap$parameters$type=="relational") {
    all.dist <- sapply(1:ncol(sommap$prototypes), function(ind) {
      sommap$prototypes%*%sommap$data[ind,]-
        0.5*diag(sommap$prototypes%*%sommap$data%*%t(sommap$prototypes))
    })
    ind.winner2 <- apply(all.dist,2,function(x) order(x)[2])
  }
  res.error <- mean(!sapply(1:nrow(sommap$data), function(x) {
      is.element(ind.winner2[x], selectNei(sommap$clustering[x],
                                           sommap$parameters$the.grid, 1))
  }))
  return(res.error)
}

quantizationError <- function(sommap) {
  if (sommap$parameters$type=="numeric") {
    quantization.error <- sum(apply((sommap$data-
                                       sommap$prototypes[sommap$clustering,])^2,
                                    1,sum))/nrow(sommap$data)
  } else if (sommap$parameters$type=="korresp") {
    norm.data <- korrespPreprocess(sommap$data)
    nr <- nrow(sommap$data)
    nc <- ncol(sommap$data)
    quantization.error <- sum(apply((norm.data[1:nr,1:nc]-
                                       sommap$prototypes[sommap$clustering[
                                         (nc+1):(nc+nr)],1:nc])^2,1,sum))
    quantization.error <- quantization.error +
      sum(apply((norm.data[(nr+1):(nr+nc),(nc+1):(nr+nc)]-
```

```
                  sommap$prototypes[sommap$clustering[1:nc],(nc+1):(nr+nc)])^2,
                1,sum))
    quantization.error <- quantization.error / (nc+nr)
  } else if (sommap$parameters$type=="relational") {
    clust.proto <- sommap$prototypes[sommap$clustering,]
    quantization.error <- clust.proto%*%sommap$data - 0.5*
      tcrossprod(diag(clust.proto%*%sommap$data%*%t(clust.proto)),
                 rep(1,ncol(sommap$data)))
    quantization.error <- sum(diag(quantization.error))/nrow(sommap$data)
  }

  quantization.error
}

# main function
quality.somRes <- function(sommap, quality.type=c("all", "quantization",
                                                  "topographic"), ...) {
  quality.type <- match.arg(quality.type)
  switch(quality.type,
         "all"=list("topographic"=topographicError(sommap),
                    "quantization"=quantizationError(sommap)),
         "topographic"=topographicError(sommap),
         "quantization"=quantizationError(sommap)
         )
}

quality <- function(sommap, quality.type,...) {
  UseMethod("quality")
}
```

A.4 superclasses.R

```
################################ subfunctions ###################################
dendro3dProcess <- function(v.ind, ind, tree, coord, mat.moy, scatter) {
  if (tree$merge[ind,v.ind]<0) {
    res <- coord[abs(tree$merge[ind,v.ind]),]
    scatter$points3d(matrix(c(res,0,res,tree$height[ind]),
                         ncol=3, byrow=TRUE), type="l")
  } else {
    res <- mat.moy[tree$merge[ind,v.ind],]
    scatter$points3d(matrix(c(res,tree$height[tree$merge[ind,v.ind]],
                           res,tree$height[ind]), ncol=3,
                         byrow=TRUE), type="l")
  }
  return(res)
}

########################### super classes V0.2 ################################
superClass.somRes <- function(sommap, method="ward", members=NULL, k=NULL,
                              h=NULL, ...) {
  if (sommap$parameters$type=="relational") {
    the.distances <- calculateProtoDist(sommap$prototypes,
                         sommap$parameters$the.grid,
                         "relational", TRUE, sommap$data)
    if (sum(the.distances<0)>0) {
      stop("Impossible to make super clustering!", call.=TRUE)
    } else the.distances <- as.dist(the.distances)
  }
  else the.distances <- dist(sommap$prototypes)^2
  hc <- hclust(the.distances, method, members)
  if (!is.null(k) || !is.null(h)) {
    sc <- cutree(hc, k, h)
    res <- list("cluster"=as.numeric(sc), "tree"=hc, "som"=sommap)
  } else {
    res <- list("tree"=hc, "som"=sommap)
```

```
  }
  class(res) <- "somSC"
  return(res)
}

superClass <- function(sommap, method, members, k, h,...) {
  UseMethod("superClass")
}

################################ S3 functions ###################################
print.somSC <- function(x, ...) {
  cat("\n   SOM Super Classes\n")
  cat("     Initial number of clusters : ", prod(x$som$parameters$the.grid$dim),
      "\n")
  if (length(x)>2) {
    cat("     Number of super clusters   : ", length(unique(x$cluster)), "\n\n")
  } else cat("     Number of super clusters not chosen yet.\n\n")
}

summary.somSC <- function(object, ...) {
  print(object)
  if (length(object)>2) {
    cat("\n  Frequency table")
    print(table(object$cluster))
    cat("\n  Clustering\n")
    print(object$cluster)
    cat("\n")
  }
}

plot.somSC <- function(x, type=c("dendrogram", "grid", "hitmap", "lines",
                                 "barplot", "boxplot", "mds", "color",
                                 "poly.dist", "pie", "graph", "dendro3d",
                                 "radar"),
                       plot.var=TRUE, plot.legend=FALSE, add.type=FALSE,
                       ...) {
  # TODO: add types "names" and "words"
  args <- list(...)
  type <- match.arg(type)

  if (type=="dendrogram") {
    args$x <- x$tree
    if (is.null(args$xlab)) args$xlab <- ""
    if (is.null(args$ylab)) args$ylab <- ""
    if (is.null(args$sub)) args$sub <- ""
    if (is.null(args$main)) args$main <- "Super-clusters dendrogram"
    if ((x$tree$method=="ward")&(plot.var)) {
      layout(matrix(c(2,2,1),ncol=3))
      Rsq <- cumsum(x$tree$height/sum(x$tree$height))
      plot(length(x$tree$height):1, Rsq, type="b", pch="+",
           xlab="Number of clusters", ylab="% of explained variance",
           main="Proportion of variance\n explained by super-clusters")
      do.call("plot", args)
    } else do.call("plot", args)
    if (length(x)>2) {
      rect.hclust(x$tree, k=max(x$cluster))
    } else warning("Impossible to plot the rectangles: no super clusters.\n",
                   call.=TRUE, immediate.=TRUE)
    par(mfrow=c(1,1), oma=c(0,0,0,0), mar=c(5, 4, 4, 2)+0.1)
  } else  if (type=="dendro3d") {
    if (length(x)==3) {
      if ((!is.null(args$col))&(length(args$col)=max(x$cluster))) {
        clust.col.pal <- args$col
        clust.col <- args$col[x$cluster]
      } else {
        if (!is.null(args$col))
          warning("Incorrect number of colors
                  (does not fit the number of super-clusters);
```

```
                using the default palette.\n", call.=TRUE, immediate.=TRUE)
      # create a color vector from RColorBrewer palette
      clust.col.pal <- brewer.pal(max(x$cluster), "Set2")
      clust.col <- clust.col.pal[x$cluster]
    }
  } else clust.col <- rep("black",prod(x$som$parameters$the.grid$dim))
  # FIX IT! maybe some more code improvements...
  x.y.coord <- x$som$parameters$the.grid$coord+0.5
  if (floor(max(x$tree$height[-which.max(x$tree$height)]))==0) {
    z.max <- max(x$tree$height[-which.max(x$tree$height)])
  } else {
    z.max <- ceiling(max(x$tree$height[-which.max(x$tree$height)]))
  }
  spt <- scatterplot3d(x=x.y.coord[,1], y=x.y.coord[,2],
                       z=rep(0,prod(x$som$parameters$the.grid$dim)),
                       zlim=c(0, z.max),
                       pch=19, color=clust.col, xlab="x", ylab="y",
                       zlab="", x.ticklabs="", y.ticklabs="")
  horiz.ticks <- matrix(NA, nrow=prod(x$som$parameters$the.grid$dim)-1, ncol=2)
  for (neuron in 1:(prod(x$som$parameters$the.grid$dim)-1)) {
    vert.ticks <- sapply(1:2, dendro3dProcess, ind=neuron, tree=x$tree,
                         coord=x.y.coord, mat.moy=horiz.ticks, scatter=spt)
    horiz.ticks[neuron,] <- rowMeans(vert.ticks)
    spt$points3d(matrix(c(vert.ticks[,1], x$tree$height[neuron],
                          vert.ticks[,2], x$tree$height[neuron]), ncol=3,
                        byrow=TRUE), type="l")
  }
} else {
  if (length(x)<3) {
    stop("No super clusters: plot unavailable.\n")
  } else {
    if ((!is.null(args$col)) & (length(args$col)=max(x$cluster))) {
      clust.col.pal <- args$col
      clust.col <- args$col[x$cluster]
    } else {
      if (!is.null(args$col))
        warning("Incorrect number of colors
                (does not fit the number of super-clusters);
                using the default palette.\n", call.=TRUE, immediate.=TRUE)
      # create a color vector from RColorBrewer palette
      clust.col.pal <- brewer.pal(max(x$cluster), "Set2")
      clust.col <- clust.col.pal[x$cluster]
    }
    if (type=="grid") {
      if (plot.legend) {
        layout(matrix(c(2,2,1),ncol=3))
        plot.new()
        legend(x="center", legend=paste("Super cluster", 1:max(x$cluster)),
               col=clust.col.pal, pch=19)
      }
      args$x <- x$som$parameters$the.grid
      args$neuron.col <- clust.col
      do.call("plot.myGrid", args)
      par(mfrow=c(1,1), oma=c(0,0,0,0), mar=c(5, 4, 4, 2)+0.1)
    } else if (type %in% c("hitmap", "lines", "barplot", "boxplot", "mds",
                           "color", "poly.dist", "pie", "graph", "radar")) {
      if ((x$som$parameters$type=="korresp") &&
            (type %in% c("boxplot", "pie", "graph")))
          stop(type, " plot is not available for 'korresp' super classes\n",
               call.=TRUE)
      if ((x$som$parameters$type=="relational") &&
            (type %in% c("boxplot", "color")))
        stop(type, " plot is not available for 'relational' super classes\n",
             call.=TRUE)

      if ((type%in%c("poly.dist", "radar"))&(plot.legend)) {
        plot.legend <- FALSE
        warning("Impossible to plot the legend with type '",type,"'.\n",
```

```
                call.=TRUE, immediate.=TRUE)
      }
      if (!(type%in%c("graph","pie", "radar"))) {
        args$col <- clust.col
      } else if (type=="graph") {
        neclust <- which(!is.na(match(1:prod(x$som$parameters$the.grid$dim),
                                      unique(x$som$clustering))))
        if (is.null(args$pie.graph)) args$pie.graph <- FALSE
        if (!args$pie.graph) {
          args$vertex.color <- clust.col[neclust]
          args$vertex.frame.color <- clust.col[neclust]
        } else {
          if (plot.legend)
            warning("Impossible to plot the legend with type '",type,"'.\n",
                    call.=TRUE, immediate.=TRUE)
          plot.legend <- FALSE
          print.title <- TRUE
          args$vertex.label <- paste("SC",x$cluster[neclust])
          args$vertex.label.color <- "black"
        }
      }
      if (plot.legend) {
        layout(matrix(c(2,2,1),ncol=3))
        plot.new()
        legend(x="center", legend=paste("Super cluster", 1:max(x$cluster)),
               col=clust.col.pal, pch=19)
        if (type%in%c("lines","barplot","boxplot","color","pie", "poly.dist",
                      "radar"))
          warning("Impossible to plot the legend with type '",type,"'.\n",
                  call.=TRUE, immediate.=TRUE)
      }
      args$x <- x$som
      if (!add.type) {
        if (type %in% c("hitmap", "boxplot")) {
          args$what <- "obs"
        } else if (type%in%c("graph","pie")) {
          args$what <- "add"
        } else args$what <- "prototypes"
      } else args$what <- "add"
      args$type <- type
      if (type=="boxplot") args$border <- clust.col
      args$the.titles <- paste("SC",x$cluster)
      if (type%in%c("pie", "radar")) {
        args$print.title <- TRUE
      } else if (type%in%c("color","poly.dist")) args$print.title <- FALSE
      do.call("plot.somRes", args)
      if (type=="color")
        text(x=x$som$parameters$the.grid$coord[,1],
             y=x$som$parameters$the.grid$coord[,2],
             labels=paste("SC",x$cluster))
      else if (type=="poly.dist")
        text(x=x$som$parameters$the.grid$coord[,1]-0.1,
             y=x$som$parameters$the.grid$coord[,2]+0.1,
             labels=paste("SC",x$cluster) )
      par(mfrow=c(1,1), oma=c(0,0,0,0), mar=c(5, 4, 4, 2)+0.1)
    } else stop("Sorry, this type is not implemented yet\n", call.=TRUE)
  }
 }
}
```

A.5 relational-test.R

```
library(SOMbrero)

iris.dist <- dist(iris[1:30,1:4], method="minkowski", diag=TRUE, upper=TRUE,
```

```
                    p=4)
rsom <- trainSOM(x.data=iris.dist, type="relational")

stopifnot(all.equal(as.vector(rowSums(rsom$prototypes)),
                      rep(1, prod(rsom$parameters$the.grid$dim))))

stopifnot(!sum(rsom$prototypes<0))
```

A.6 Script démo : produire le logo de SOMbrero

```
library(scatterplot3d)
library(RColorBrewer)

n <- 250
x <- seq(-10, 10, length=n)
y <- x
f <- function(x,y) {
  r <- sqrt(x^2+y^2)
  10 * sin(r)/r
}
z <- outer(x, y, f)
z[is.na(z)] <- 1
x <- matrix(x,nr=n,nc=n)
y <- matrix(y,nr=n,nc=n,byrow=T)
sombrero <- data.frame(x=as.vector(x),y=as.vector(y),z=as.vector(z))
scatterplot3d(sombrero,color=brewer.pal(7,"Set2")[cut(1:nrow(sombrero),7,label=FALSE)],,
    pch=".")

## numeric SOM example with demo(numeric)
## korresp SOM example (for contingency tables) with demo(korresp)
## relational SOM example (for dissimilarity data) with demo(relational)
```

Annexe B

Documentations

B.1 Exemple de documentation R : *quality.Rd*

```
\name{quality}
\alias{quality.somRes}
\alias{quality}

\title{Compute SOM algorithm quality criteria}
\description{The \code{quality} function computes several quality criteria for
the result of a SOM algorithm.}
\usage{
\method{quality}{somRes}(sommap,
        quality.type=c("all", "quantization", "topographic"), \ldots)
}

\arguments{
  \item{sommap}{A \code{somRes} object (see \code{\link{trainSOM}} for
  details).}
  \item{quality.type}{The quality type to compute. Two types are implemented:
  \code{quantization} and \code{topographic}. The output of the function is
  one those or both of them using the option \code{"all"}. Default value is the
  latter.}
  \item{\ldots}{Not used.}
}

\value{The \code{quality} function returns either a numeric value (if only one
type is computed) or a list a numeric values (if all types are computed)

The quantization error calculates the mean distance between the sample vectors
and their respective cluster prototypes. It is a decreasing function of the size
of the map.

The topographic error is the simplest of the topology preservation measure: it
calculates the ratio of sample vectors for which the second best matching unit
is in the direct neighborhood of the best matching unit.
}

\author{Laura Bendhaiba \email{laurabendhaiba@gmail.com}\cr
Madalina Olteanu \email{madalina.olteanu@univ-paris1.fr}\cr
Nathalie Villa-Vialaneix \email{nathalie.villa@univ-paris1.fr}
}

\references{
Polzlbauer, G. (2004) Survey and comparison of quality measures for
self-organizing maps. In: \emph{Proceedings of the Fifth Workshop on Data
Analysis (WDA'04), Paralic, J., Polzlbauer, G., Rauber, A. (eds)} Sliezsky dom,
Vysoke Tatry, Slovakia: Elfa Academic Press, 67--82.
}
```

```
\seealso{\code{\link{trainSOM}}, \code{\link{plot.somRes}}}

\examples{
my.som <- trainSOM(x.data=iris[,1:4])
quality(my.som, quality.type="all")
quality(my.som, quality.type="topographic")
}
\keyword{cluster}
```

B.2 Exemple de rendu de documentation utilisateur (vignette)

Self-Organizing Map for contingency tables

Laura BENDHAIBA, Madalina OLTEANU, Nathalie VILLA-VIALANEIX

Basic package description

To be able to run the SOM algorithm, you have to load the package called `SOMbrero`. The function used to run it is called `trainSOM()` and is detailed below.

This documentation only considers the case of contingency tables.

Arguments

The `trainSOM` function has several arguments, but only the first one is required. This argument is `x.data` which is the dataset used to train the SOM. In this documentation, it is passed to the function as a matrix or a data frame. This set must be a contingency table, i.e., it must contain either 0 or positive integers. Column and row names must be supplied.

The other arguments are the same than the arguments passed to the `initSOM` function (they are parameters defining the algorithm, see `help(initSOM)` for further details).

Outputs

The `trainSOM` function returns an object of class `somRes` (see `help(trainSOM)` for further details on this class).

Case study: the `presidentielles2002` data set

The `presidentielles2002` data set provides the number of votes at the first round of the 2002 French presidential election for each of the 16 candidates in all of the 106 French administrative districts called “departements”. Further details about this data set and the 2002 French presidential election are given with `help(presidentielles2002)`.

```
data(presidentielles2002)
apply(presidentielles2002, 2, sum)
```

```
##      MEGRET      LEPAGE  GLUCKSTEIN      BAYROU
CHIRAC      LE_PEN
##      667043      535875      132696     1949219     5666021
4804772
##     TAUBIRA SAINT_JOSSE      MAMERE      JOSPIN
BOUTIN         HUE
##      660515     1204801     1495774     4610267
339157      960548
## CHEVENEMENT     MADELIN   LAGUILLER  BESANCENOT
##     1518568     1113551     1630118     1210562
```

(the two candidates that ran the second round of the election were Jacques Chirac and the far-right candidate Jean-Marie Le Pen)

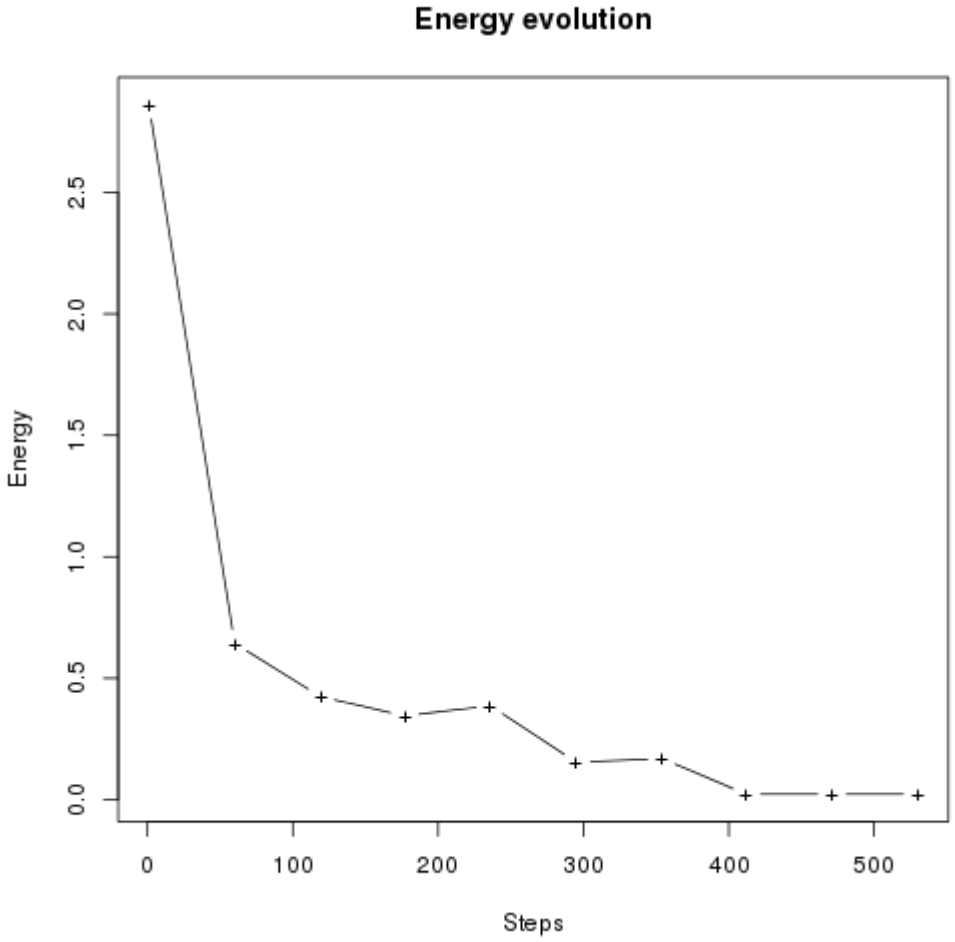
Training the SOM

```
set.seed(4031719)
korresp.som <- trainSOM(x.data = presidentielles2002, dimension =
c(8, 8), type = "korresp",
    scaling = "chi2", nb.save = 10)
korresp.som
```

```
##       Self-Organizing Map object...
##          online learning, type: korresp
##          8 x 8 grid with square topology
```

As the energy is registered during the intermediate backups, we can have a look at its evolution

```
plot(korresp.som, what = "energy")
```



which is stabilized during the last 100 iterations.

Resulting clustering

The clustering component contains the final classification of the dataset. As both row and column variables are classified, the length of the resulting vector is equal to the sum of the number of rows and the number of columns.

NB: The clustering component shows first the column variables (here, the candidates) and then the row variables (here, the departements).

```
korresp.som$clustering
```

```
##                    MEGRET                     LEPAGE
GLUCKSTEIN
##                         8
22                         51
##                    BAYROU
CHIRAC                     LE_PEN
##                         18
31                         23
##                   TAUBIRA
SAINT_JOSSE                      MAMERE
##                         42
64                          6
##                    JOSPIN
BOUTIN                           HUE
##                          5
9                         62
##               CHEVENEMENT
MADELIN                  LAGUILLER
##                         22
22                         51
##                 BESANCENOT
ain                      aisne
##                         51
24                         53
##                     allier  alpes_de_haute_provence
hautes_alpes
##                         54
55                         32
##           alpes_maritimes
ardeche                   ardennes
##                         24
32                         53
##                     ariege
aube                       aude
##                         48
22                         55
##                    aveyron
bouches_du_rhone                  calvados
##                         32
53                         32
##                     cantal                  charente
charente_maritime
##                         40
32                         48
##                       cher
correze                  corse_sud
##                         54
```

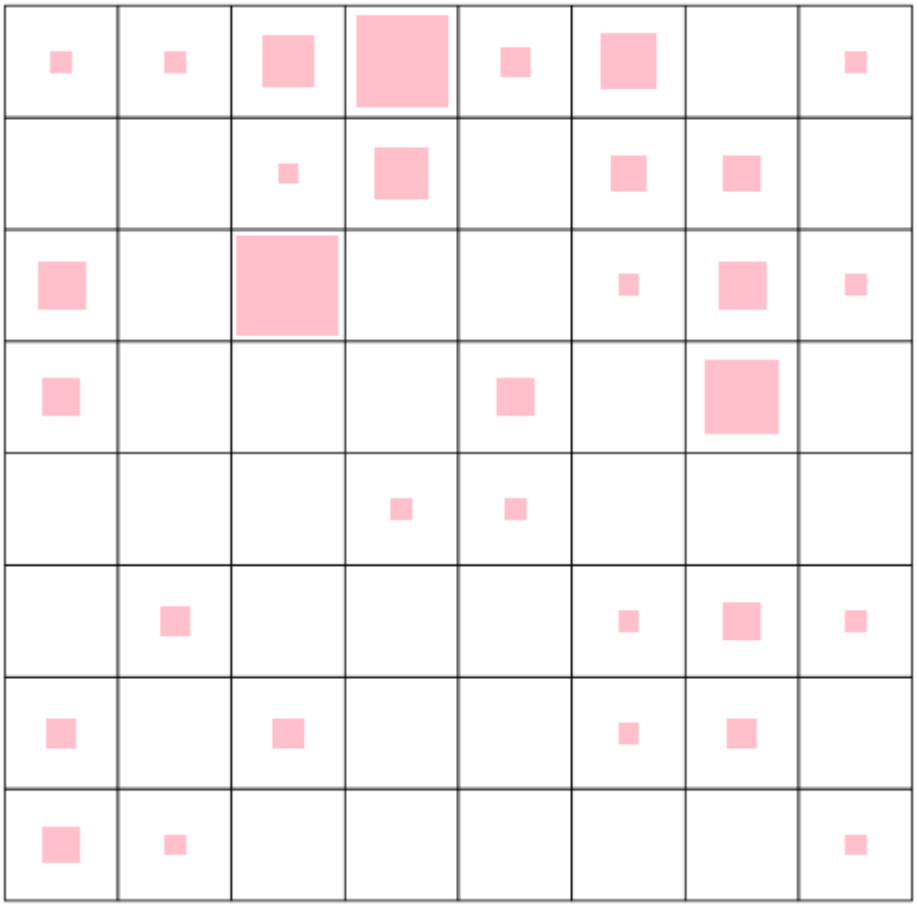
```
46                        32
##               haute_corse                 cote_d'or
cotes_d'armor
##                        47
22                        32
##                     creuse
dordogne                       doubs
##                        47
47                        22
##                      drome                      eure
eure_et_loir
##                        22
22                        22
##                  finistere                      gard
haute_garonne
##                        31
53                         6
##                       gers
gironde                    herault
##                        48
32                        53
##           ille_et_vilaine                     indre
indre_et_loire_
##                        31
32                        22
##                     isere
jura                     landes
##                          6
22                        48
##              loir_et_cher                     loire
haute_loire
##                        32
53                        22
##          loire_atlantique
loiret                         lot
##                        31
22                        48
##           lot_et_garonne_                    lozere
maine_et_loire_
##                        55
32                        31
##                     manche                     marne
haute_marne
##                        32
22                        24
##                    mayenne
meurthe_et_moselle                     meuse
```

```
##                       31
53                       22
##                  morbihan
moselle                   nievre
##                       22
53                       54
##                     nord
oise                       orne
##                       53
53                       32
##             pas_de_calais              puy_de_dome
pyrenees_atlantiques
##                       54
22                       40
##          hautes_pyrenees
pyrenees_orientales                bas_rhin
##                       48
53                       18
##                 haut_rhin                     rhone
haute_saone
##                        1
1                       24
##          saone_et_loire_
sarthe                   savoie
##                       22
22                       22
##              haute_savoie                     paris
seine_maritime_
##                        1
11                       53
##          seine_et_marne_                 yvelines
deux_sevres
##                        6
2                       32
##                    somme                      tarn
tarn_et_garonne
##                       48
32                       32
##                      var
vaucluse                   vendee
##                       24
24                       32
##                    vienne
haute_vienne                     vosges
##                       32
54                       22
##                    yonne
```

```
territoire_de_belfort                  essonne
##                        22
16                          5
##          hauts_de_seine_        seine_saint-denis
val_de_marne
##                         2
59                          5
##               val_d'oise               guadeloupe
martinique
##                         6
50                         50
##                    guyane
la_reunion                   mayotte
##                         57
28                         36
##       nouvelle_caledonie      polynesie_francaise
saint_pierre_et_miquelon
##                         37
37                         43
##         wallis_et_futuna   francais_de_l'etranger
##                         37                         11
```

The resulting distribution of the clustering on the map can also be visualized by a hitmap:

```
plot(korresp.som, what = "obs", type = "hitmap")
```



For a more precise view, "names" plot is implemented: it prints, in each neuron, the names of the variables assigned to it ; in the korresp SOM, both row and column variable names are printed.

```
plot(korresp.som, what = "obs", type = "names", scale = c(1.5,
0.5))
```

Annexe C

Rencontres R 2013 : article soumis

SOMbrero : Cartes auto-organisatrices stochastiques pour l'intégration de données décrites par des tableaux de dissimilarités

Laura Bendhaïba[a], **Madalina Olteanu**[a] et **Nathalie Villa-Vialaneix**[a,b]

[a]SAMM, Université Paris 1
F-75634 Paris - France
laurabendhaiba@gmail.com
{madalina.olteanu,nathalie.villa}@univ-paris1.fr

[b]INRA, UR875, MIAT
F-31326 Castanet Tolosan - France

Mots clefs : cartes auto-organisatrices, dissimilarités, graphes, classification, visualisation

Dans de nombreuses situations réelles, les individus sont décrits par des jeux de données multiples qui ne sont pas nécessairement de simples tableaux numériques mais peuvent être des données complexes (graphes, variables qualitatives, texte...). Un cas typique est celui des graphes étiquetés dans lequel les individus (les sommets du graphe) sont décrits à la fois par leurs relations les uns aux autres mais aussi par des attributs de natures diverses. Dans [5, 2], nous avons proposé d'utiliser des cartes auto-organisatrices [1] pour combiner classification et visualisation en projetant les individus étudiés sur une grille de faible dimension. Notre approche permet de traiter des données non numériques par le biais de noyaux ou de dissimilarités, et est basée sur une version stochastique de l'apprentissage de cartes auto-organisées, comme décrit dans [4, 3]. Les différentes dissimilarités sont combinées et la combinaison est optimisée au cours de l'apprentissage de la carte.

Nous avons testé notre approche sur un jeu de données simulé : dans celui-ci, les observations sont décrites par un graphe séparé en deux groupes denses de sommets (figure 1, en haut à gauche), les sommets étant étiquetés par des valeurs numériques de $\mathbb{R}^2$ tirées selon deux Gaussiennes (figure 1) ainsi que par un facteur à deux niveaux. Seules les trois informations permettent de retrouver les 8 groupes de sommets, représentés par 8 couleurs différentes sur la figure 1. La combinaison des trois informations sous la forme de trois tableaux de dissimilarités (longueur du plus court chemin entre deux sommets pour le graphe, distance euclidienne pour les étiquettes numériques et distance de Dice pour les facteurs) permet de retrouver les huit groupes initiaux avec une bonne précision et de bien les organiser sur la carte (figure 1, en bas à droite). L'apprentissage adaptatif des distances donne un poids prépondérant à la dissimilarité basée sur la valeur du facteur qui est la seule valeur non bruitée (figure 1).

La méthodologie proposée est en voie d'implémentation dans un package R appelé **SOMbrero**. La version 0.1 du package, disponible depuis mars 2013 propose l'implémentation de l'algorithme de cartes auto-organisatrices pour des données numériques simples ainsi que diverses fonctionnalités permettant l'interprétation (fonctionnalité graphique pour visualiser les niveaux des diverses variables, les valeurs des prototypes de la carte...). Le package n'est pas encore disponible sur le CRAN mais peut être téléchargé à http://tuxette.nathalievilla.org/?p=1099&lang=en (sources et compilation windows).

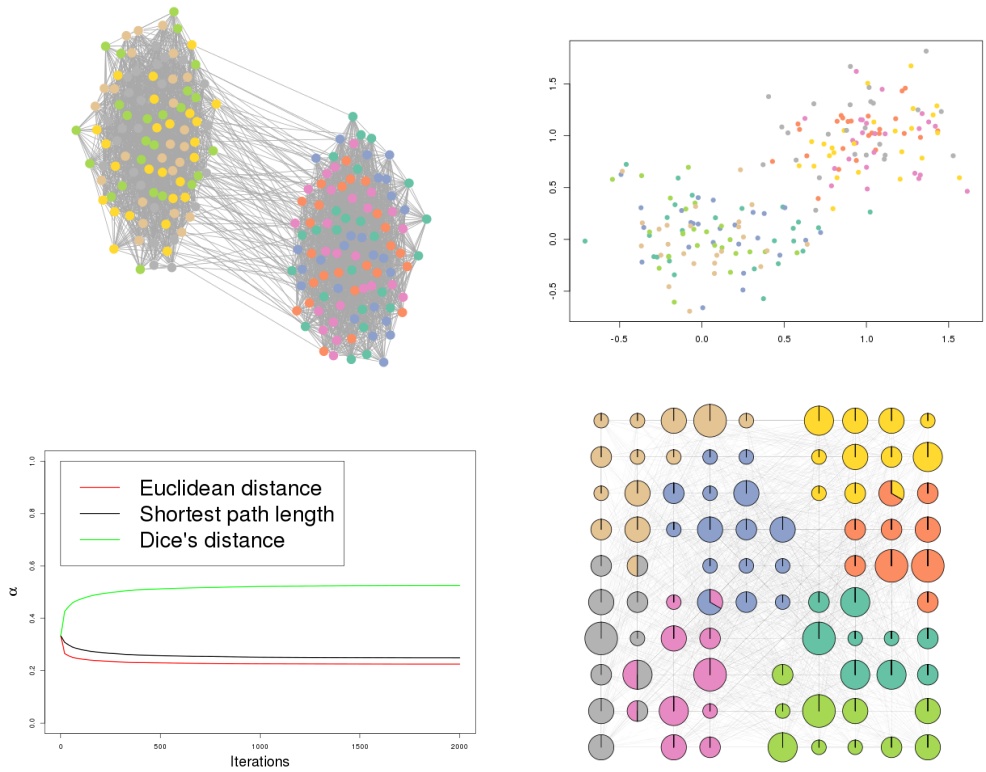


Figure 1: Données simulées : graphes et valeurs des étiquettes numériques des sommets (en haut à gauche et à droite). Évolution des poids des diverses dissimilarités (en bas à gauche). Carte finale obtenue (en bas à droite, les couleurs représentent les classes initiales, les aires des disques sont proportionnelles au nombre d'observations de la classe)

References

[1] T. Kohonen. *Self-Organizing Maps, 3rd Edition*, volume 30. Springer, Berlin, Heidelberg, New York, 2001.

[2] M. Olteanu, N. Villa-Vialaneix, and C. Cierco-Ayrolles. Multiple kernel self-organizing maps. In M. Verleysen, editor, *XXIst European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, Bruges, Belgium, 2013. d-side publications. Forthcoming.

[3] M. Olteanu, N. Villa-Vialaneix, and M. Cottrell. On-line relational som for dissimilarity data. In P.A. Estevez, J. Principe, P. Zegers, and G. Barreto, editors, *Advances in Self-Organizing Maps (Proceedings of WSOM 2012)*, volume 198 of *AISC (Advances in Intelligent Systems and Computing)*, pages 13–22, Santiago, Chile, 2012. Springer Verlag, Berlin, Heidelberg.

[4] N. Villa and F. Rossi. A comparison between dissimilarity SOM and kernel SOM for clustering the vertices of a graph. In *6th International Workshop on Self-Organizing Maps (WSOM)*, Bielefield, Germany, 2007. Neuroinformatics Group, Bielefield University.

[5] N. Villa-Vialaneix, M. Olteanu, and C. Cierco-Ayrolles. Carte auto-organisatrice pour graphes étiquetés. In *Actes des Ateliers FGG (Fouille de Grands Graphes), colloque EGC (Extraction et Gestion de Connaissances)*, Toulouse, France, 2013.