Projet 1 SY09

Yann MAILLET - Capucine PRUDHOMME

May 4, 2018

1 Cuisine

1.1 Etude exploratoire des données

Le jeu de données est un tableau de 26 observations (individus) sur 50 variables. Parmi ces variables, 49 sont quantitatives et correspondent à des ingrédients (nous supposons que les valeurs sont le poids en kilogrammes de l'ingrédient dans la recette. La variable qualitative correspond à l'origine de la recette. Pour chaque observation i, on a la quantité de l'ingrédient j (pour j de 1 à 49) dans la case d'indice ij.

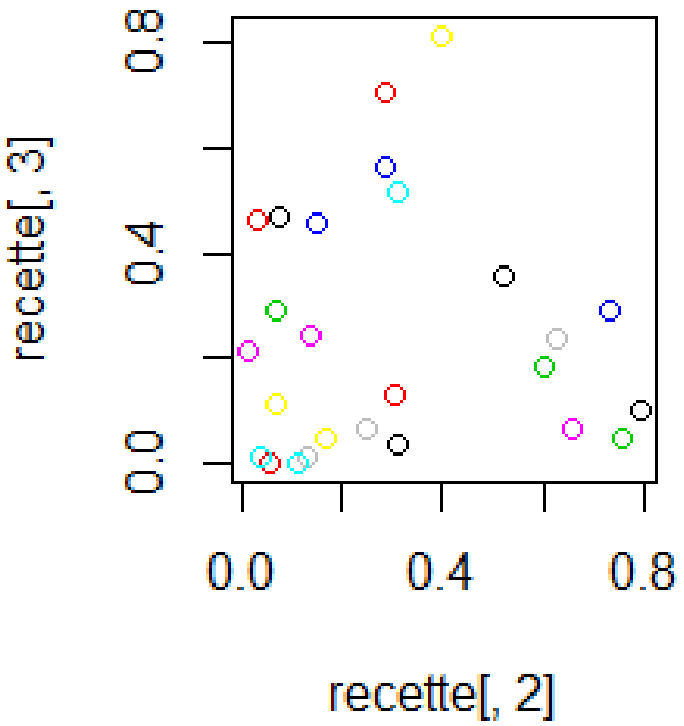

Figure 1: Proportion de cayenne par rapport à l'huile d'olive.

Il est possible de cette manière de comparer les proportions respectives des ingrédients 2 à 2, et d'étudier l'influence des origines sur ces proportions et sur les ingrédients utilisés. De plus, à l'aide d'un rowSums, nous pouvons remarquer que le nombre moyen d'ingrédients par recettes

est entre 44 et 45, et que le poids moyen d'une recette est de 6 kilos.
En étudiant le tableau de covariances, on peut remarquer que certains ingrédients sont fortement corrélés, comme par exemple la sauce soja et le turmeric, la sauce soja et le riz, ou encore le beurre et la crème. Ceci peut être lié à l'origine des produits, et leur utilisation dans différentes régions du monde. Sur les graphs ci-dessous nous pouvons voir les résultats de la visualisation de la corrélation entre éléments très peu ou très corrélés.

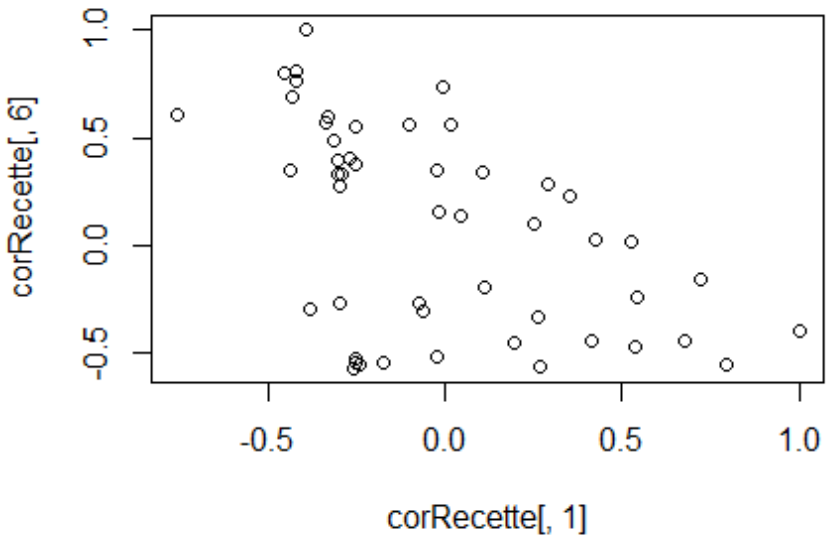

Figure 2: Corrélation entre l'huile d'olive et le gingembre.

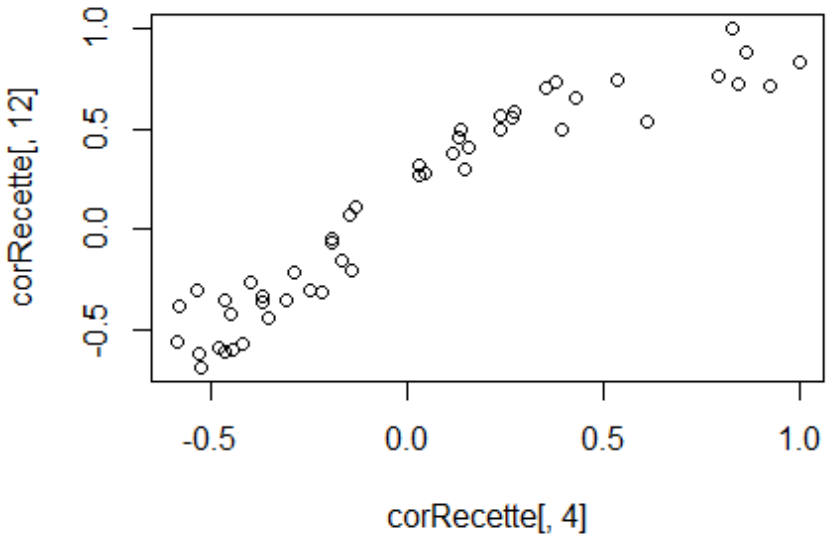

Figure 3: Corrélation entre le riz et la sauce soja.

1.2 ACP

Nous avons ensuite pu étudier les données par l'analyse des composantes principales. Nous remarquons qu'avec 5 composantes principales nous atteignons 87% de l'information, et 90% avec 6. (voir annexes)

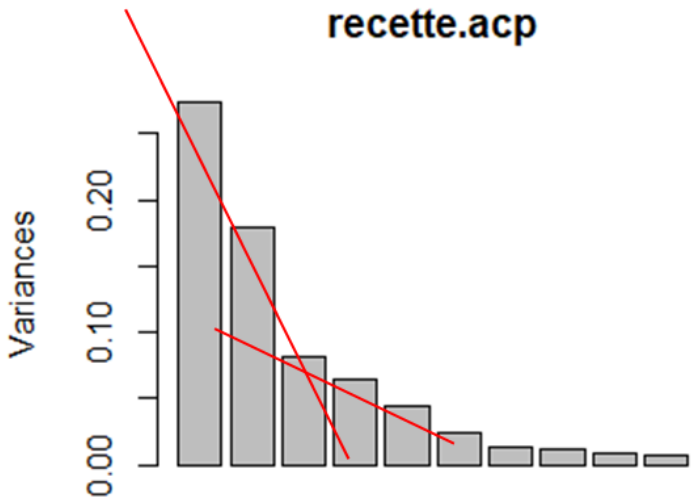

Figure 4: Graphique de perte d'inertie.

Grace à ce graphique représentant la perte d'inertie selon le nombre de composantes principales, nous pouvons utiliser la méthode du coude afin de déterminer le nombre de composantes principales à garder. Nous voyons qu'il y a une cassure du coude au niveau de la troisième composante principale (ainsi 72% de l'information est sauvegardée).
Les deux premières composantes principales permettent de représenter 61% de l'information. Voici ci-dessous la représentation des données dans le premier plan factoriel, avec les deux premières composantes principales. Nous avons décidé de représenter les origines pour pouvoir, à vue d'œil, estimer l'impact de celles-ci sur les données obtenues.

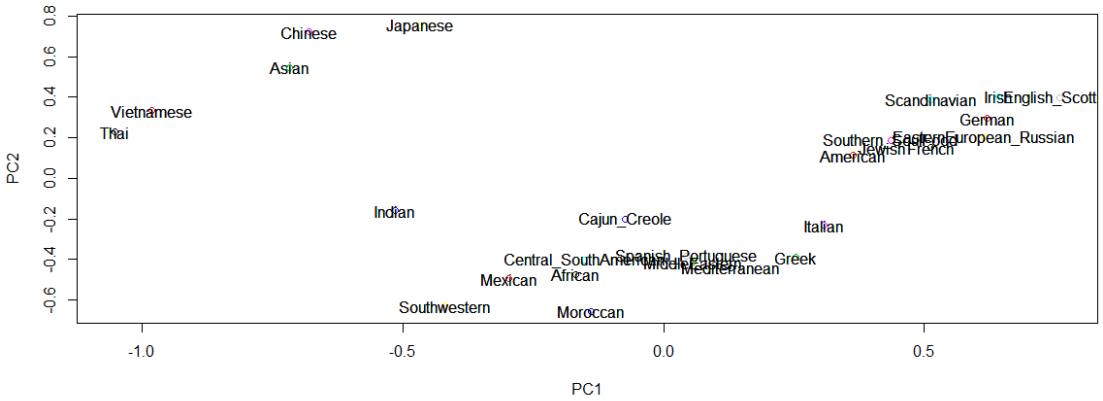

Figure 5: Représentation des données dans le premier plan factoriel.

Sur ce graphique, nous pouvons remarquer que les données tendent à se répartir en trois groupes.

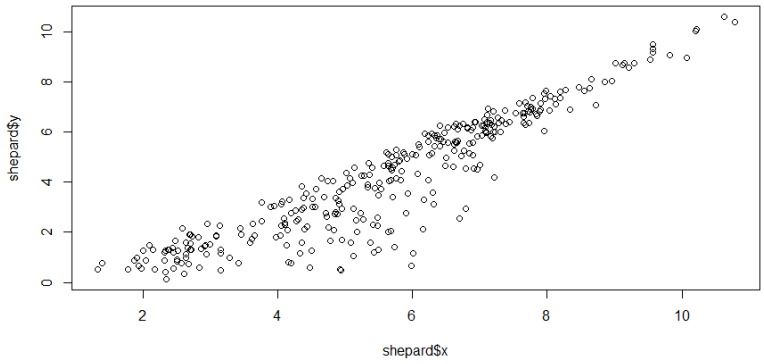

Figure 6: Diagramme de Shepard.

Voici le diagramme de Shepard correspondant à ces données.

1.3 CAH

Nous faisons une classification ascendante hiérarchique en utilisant la distance de Manhattan pour notre matrice de distance. Ensuite, nous utilisons le critère d'agrégation Ward.D2 qui prend le carré de la distance, afin de créer un dendrogramme. Nous pouvons de nouveau remarquer la tendance des données à se répartir en 3 groupes distincts.

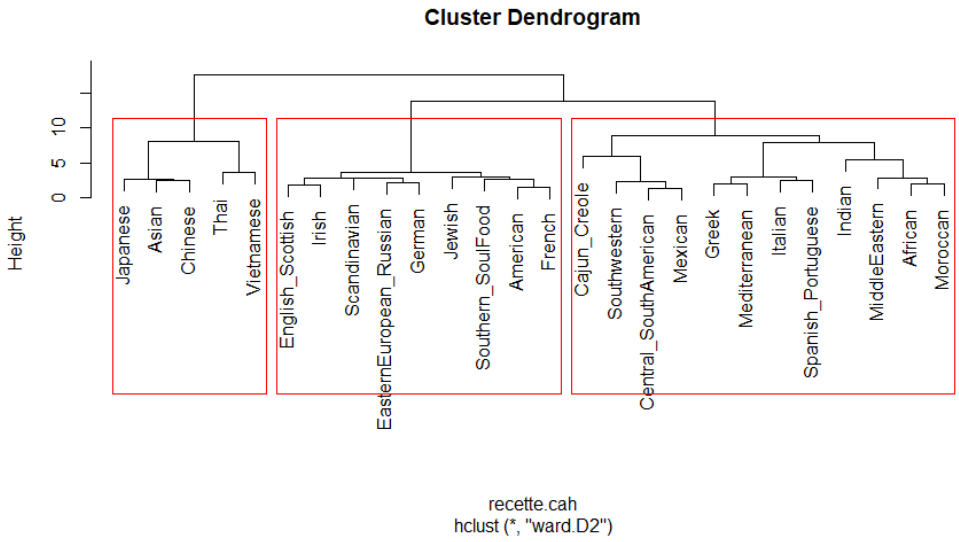

Figure 7: Dendrogramme de la CAH sur les données recettes.

1.4 Algorithme des Kmeans

Afin de déterminer le nombre de classes le plus adéquat, nous mesurons l'inertie intraclasse pour différents K. Nous affichons ce graphe et nous appliquons dessus la méthode du coude.

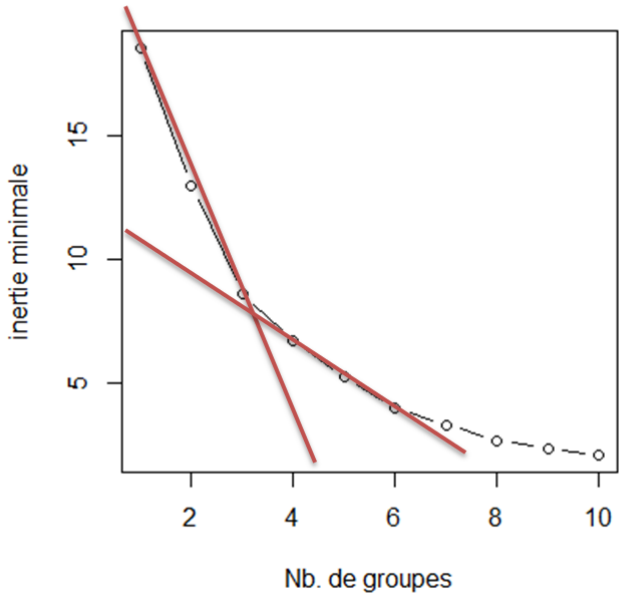

Figure 8: Graphique montrant l'inertie intraclasse en fonction du nombre de classes.

Pour appliquer les kmeans, nous avons commencé par centrer le tableau (pas de nécessité de la réduire puisque les valeurs sont comparables), puis nous avons appliqué 10 fois l'algorithme pour k=3 classes.
Nous obtenons 3 groupes de respectivement 12, 5 et 9 éléments.

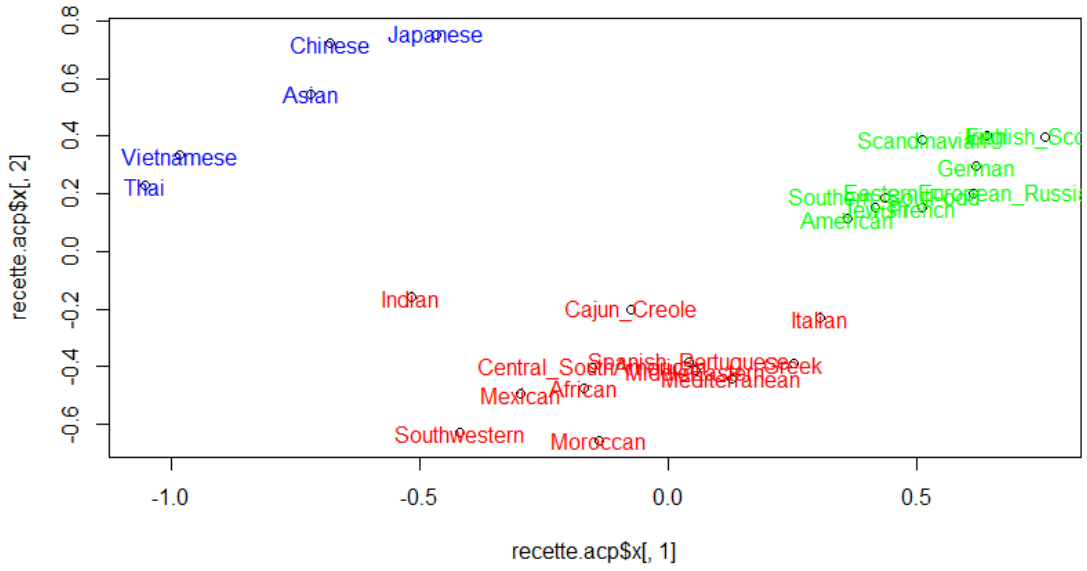

Figure 9: Représentation des données dans le premier plan factoriel, avec une couleur par groupe.

Sur le graphique ci-dessous, représentant les données dans la couleur correspondant à leur groupe (groupes calculés grâce à l'algorithme des Kmeans), nous apercevons un lien géographique avec la répartition en groupes. En effet, ces groupes peuvent être interprétés comme :

- Pays chauds (Afrique, Amérique centrale et latine, pays méditerranéens) en rouge
- Pays asiatiques en bleu
- Pays tempérés, en Europe et Amérique du Nord en vert

Cependant, nous avons décidé d'utiliser la méthode du coude pour déterminer le nombre de classes. Mais nous aurions pu choisir une autre méthode. En effet, il aurait pu être intéressant de prendre une classe en plus afin de séparer les données de la classe rouge, pour obtenir 2 classes distinctes parmi les pays chauds : les pays méditerranéens/ Afrique, et les pays d'Amérique latine.

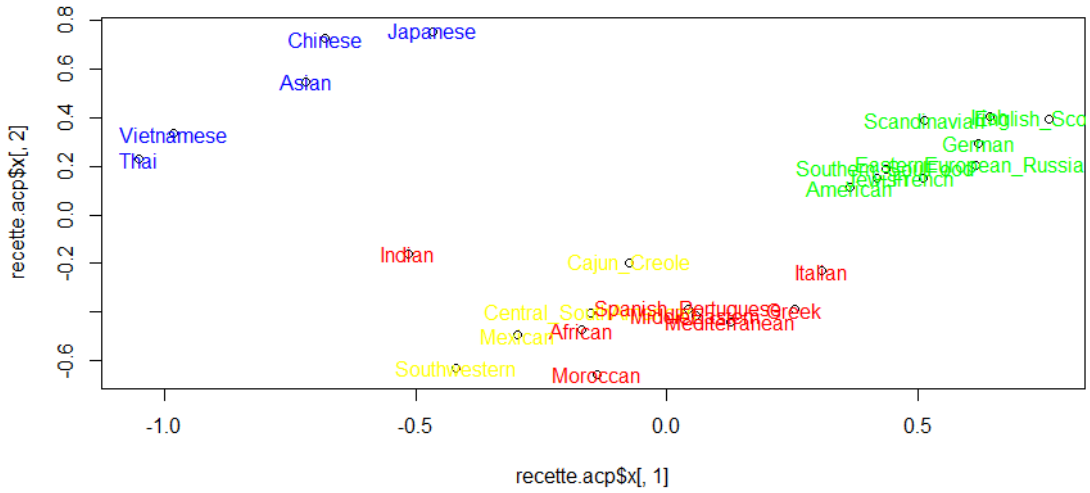

Figure 10: Kmeans avec 4 classes.

Nous voyons bien une distinction de couleur entre les pays méditerranéens et Afrique, et les pays créoles et Amérique latine. Cependant, ces classes sont assez confondues sur le premier plan factoriel, nous décidons de ne conserver que 3 classes même si nous perdons un peu de précision dans la classification.

1.5 Analyse descriptive recettes-echant

Le fichier de données recettes-echant.data est un fichier de 2000 observations sur 51 variables. Il y a de nouveau 1 colonne qualitative (le pays d'origine) et 50 variables quantitatives. Celles-ci sont en binaire : chaque variable représente un ingrédient, et chaque individu est en fait une recette. Pour chaque ingrédient et chaque recette, si l'ingrédient est présent alors la valeur est 1, sinon c'est 0 (codage sur la présence ou l'absence de l'ingrédient dans la recette).
En calculant le nombre moyen d'ingrédients par recette, nous trouvons 6. Dans ce jeu de données, les variables quantitatives sont très peu corrélées.

1.6 ACP Recette-echant

En effectuant une ACP sur ce jeu de données, nous nous rendons compte qu'il y a une très faible perte d'inertie en augmentant le nombre de composantes principales. La première composante principale ne représente que 13% de l'information.

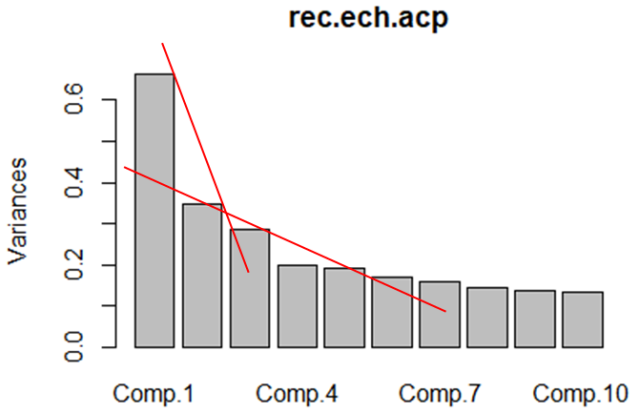

Figure 11: Graphique montrant la perte d'inertie en fonction du nombre de CP.

Cependant, en appliquant la méthode du coude, nous nous apercevons que nous pouvons nous contenter de 2 composantes principales, même si elles ne représentent que 20% de l'information.

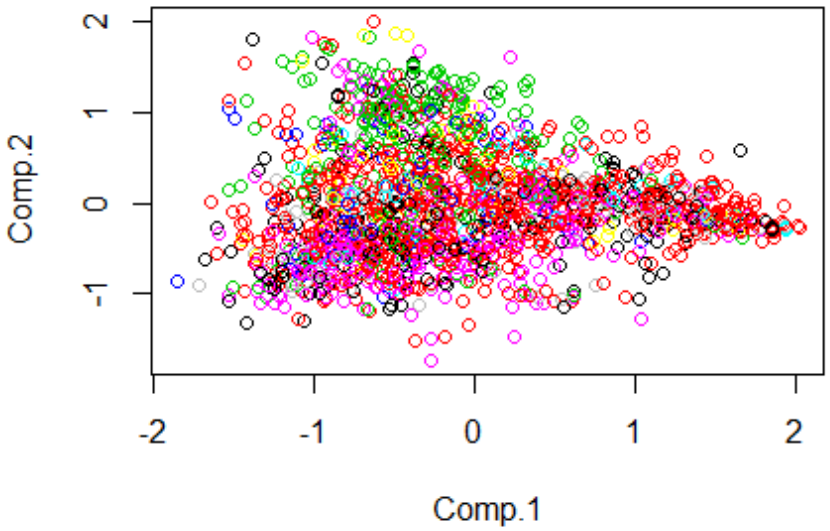

Figure 12: Représentation dans le premier champ factoriel.

1.7 Matrice de similarité

Afin de regrouper les ingrédients en groupes avec des caractéristiques homogènes, nous décidons de grouper les ingrédients par origine. Pour cela, nous utilisons la bibliothèque plyr afin de faire la moyenne des ingrédients par recette, et grouper ceux-ci. Nous transposons la matrice

de dissimilarité afin d'obtenir les ingrédients en individus et les origines en variables.
Nous pouvons ensuite effectuer une ACP sur cette matrice. Voici la représentation des données dans le premier plan factoriel suite à une ACP.

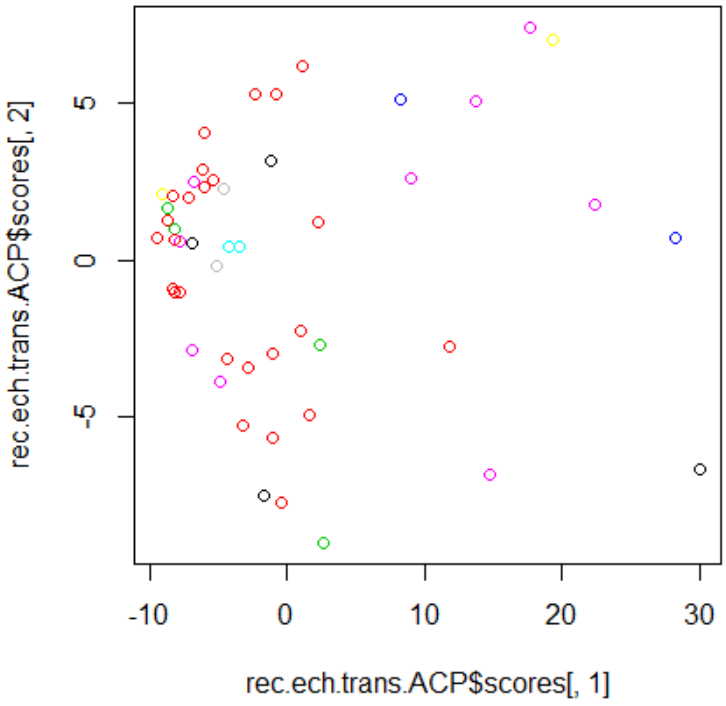

Figure 13: Représentation des données dans le premier champ factoriel.

Les deux premières composantes principales représentent à elles seules 85% de l'information. Nous pouvons déjà voir sur ce plot que les données pourraient être classées plus facilement que précédemment.

1.8 CAH recette-echant

Afin de mettre en évidence la possible séparation des ingrédients en groupes, nous effectuons une CAH avec la méthode ward.D2. Avant cela, nous appliquons la même fonction que précédemment pour déterminer grâce à la méthode du coude le nombre de classe le plus judicieux à avoir.

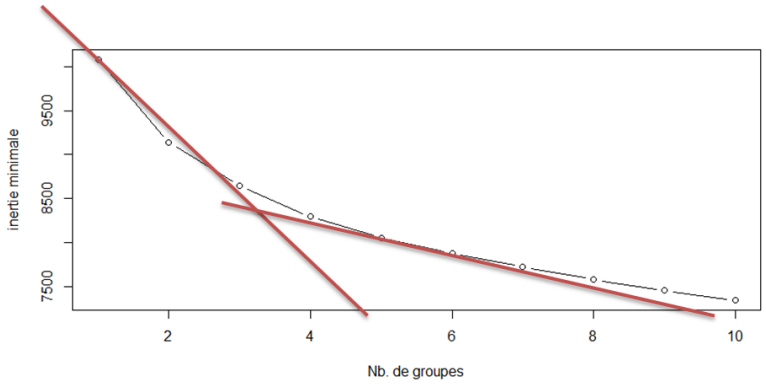

Figure 14: Représentation de l'inertie minimale en fonction du nombre de groupes.

Nous pouvons voir qu'un nombre de groupe intéressant serait 4. Bien sûr, ceci est contestable, mais un tel nombre de groupes sépare les ingrédients en des groupes de taille pas trop faible, et reste dans une classification assez générale.

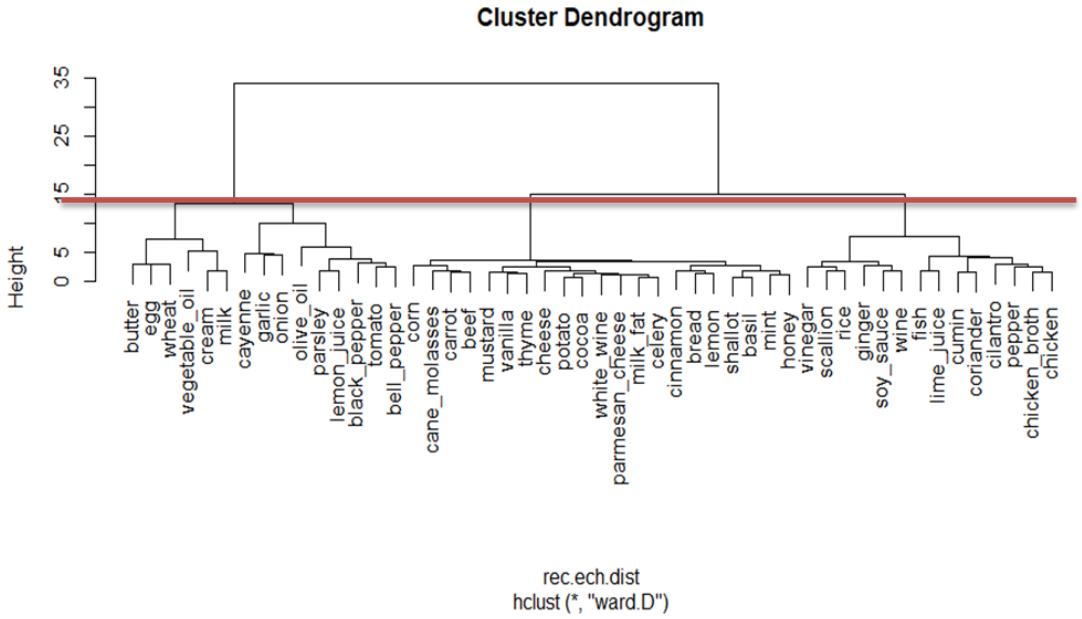

Figure 15: Dendrogramme de la CAH sur recette-echant.

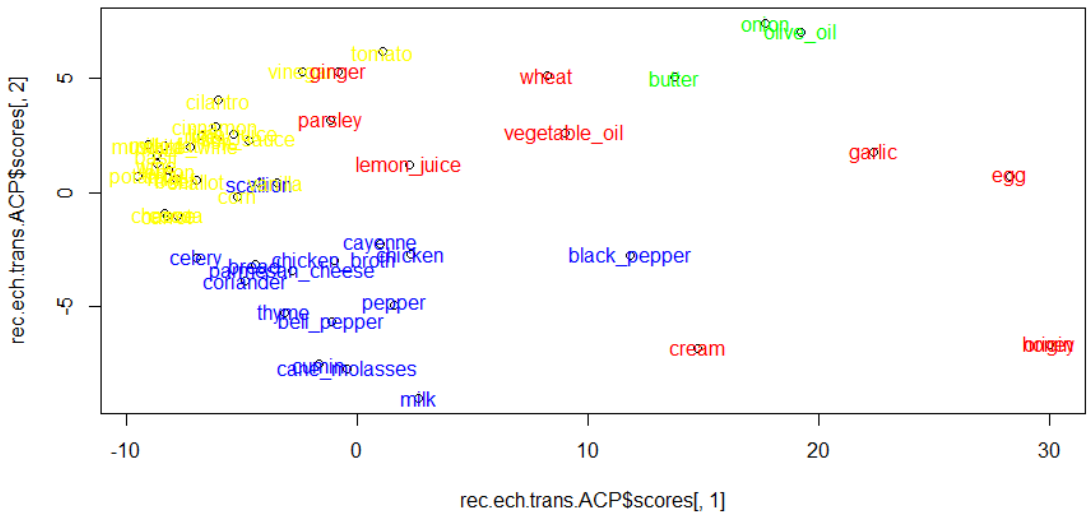

Figure 16: Représentation sur le premier plan factoriel des ingrédients répartit selon les groupes obtenus par la CAH.

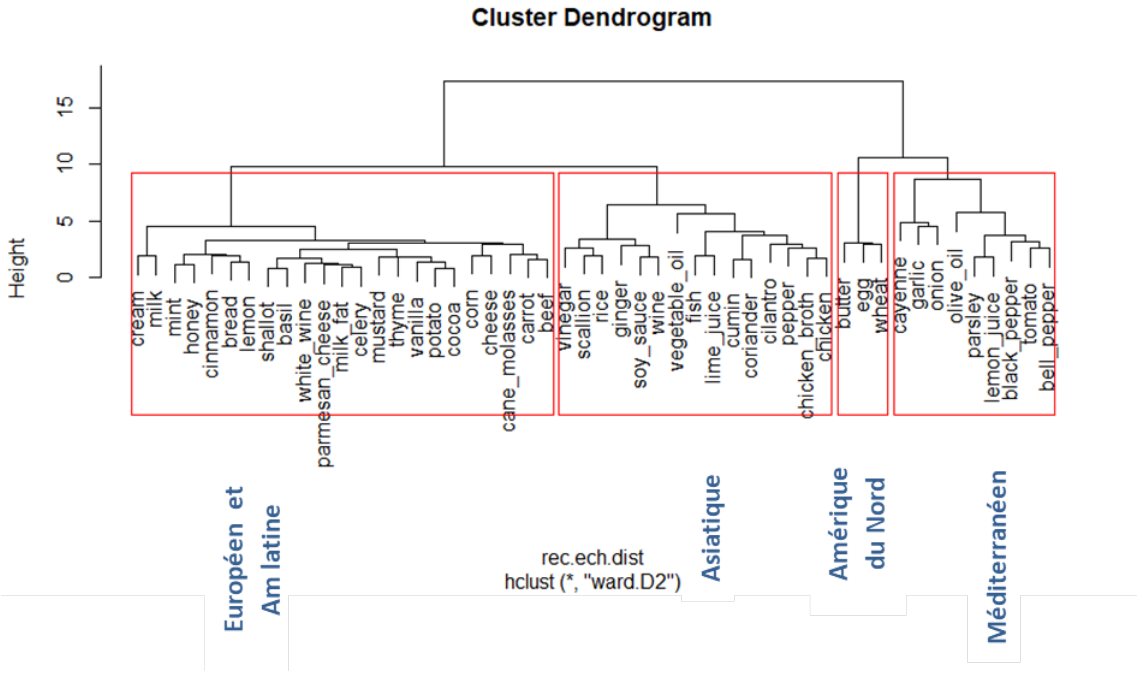

Figure 17: Dendrogramme suite à la transformation des données recette.echant.

1.9 K-medoïdes

En appliquant la méthode des médoïdes avec k=4, nous obtenons quatre valeurs correspondant à des ingrédients : "Garlic", "Chicken", "Black_pepper" et "milk_fat". Nous regardons

ensuite les ingrédients présents dans la même classe que chacun de ces 4 ingrédients sur le dendrogramme. Voici les 4 groupes obtenus :

- Gr 1: **milk_fat**, cream, milk, cinnamon, bread, white_wine, corn, cocoa, potato, vanilla, carrot, beef, cane_molasses, mustard, etc.
- Gr 2: **chicken**, vinegar, scallion, rice, ginger,soy_sauce, fish, lime_juice, coriander, etc.
- Gr 3: butter, egg, wheat
- Gr 4: **garlic, black_pepper**, onion, parsley, olive_oil, tomato, bell_pepper, etc

Nous voyons que la répartition des k-medoïdes ne colle pas parfaitement avec la répartition de la question 7, puisque les groupes 1 et 2 ont bien chacun un médoïde dans leur groupe. Mais le groupe 3 n'en a aucun tandis que le groupe 4 en a 2.
Nous pouvons tout de même paralléliser ces résultats avec ceux de la première étude. Les ingrédients du groupe 1 sont très caractéristiques des recettes d'Europe et d'Amérique latine. Le groupe 2 a des ingrédients types de la cuisine asiatique (en ce sens cela se rapproche de la première classification). Le groupe 3 n'a que 3 ingrédients, mais qui sont très utilisés en Amérique du Nord, et Europe. Enfin, le groupe 4 a des ingrédients à base d'herbes, d'huile d'olive et de légumes du soleil, la base de la cuisine méditerranéenne.

2 Classification par K-means avec distance adaptative

2.1 Données synthétiques

Les données SynthN sont un ensemble de 200 individus ayant deux coordonnées x1 et x2, et une variable qualitative z valant 1 ou 2 (nous supposons que c'est la classe réelle des variables) réparti en 2 classes de même taille.

2.1.1 Synth1 :

Nous affectons sur ce jeu de données préalablement centré une première classification avec l'algorithme des Kmeans, et nous représentons la perte d'inertie en fonction du nombre de classes K pour k de 1 à 10.

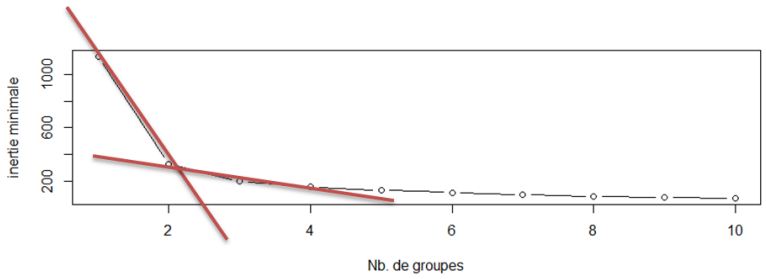

Figure 18: Graphique montrant la perte d'inertie en fonction du nombre de classes.

Nous pouvons ainsi appliquer la méthode du coude pour déterminer le nombre de classes le plus approprié, avec cet algorithme des kmeans. Nous trouvons que 2 est un bon nombre, ce qui donne deux classes de 101 et 99 éléments. En appliquant la méthode adjustedRandIndex, nous trouvons une valeur de 0.9799995 ce qui indique une forte similarité avec la classification réelle (variable qualitative z).

Nous faisons de même avec ce jeu de données, mais cette fois ci en utilisant l'algorithme des K-means avec distance adaptative. En appliquant la méthode du coude, nous remarquons que une fois encore c'est le nombre 2 qui parait le plus intéressant. Cependant, la courbe a un plateau de k=2 à k=5. Ceci signifie qu'il y a très peu de perte de distance en augmentant le nombre de classes, la courbe est très différente de la précédente. En comparant avec les classes réelles pour k=2, nous avons une similarité de 0.9799995. Les deux algorithmes sont donc très adaptés à ce jeu de données, tout du moins ils donnent une classification proche de la classification réelle.

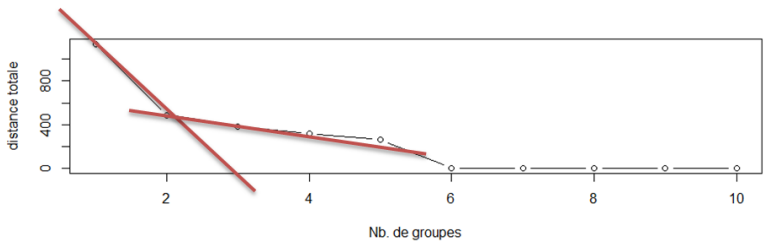

Figure 19: Graphique montrant la perte de distance en fonction du nombre de classes.

K	1	2	3	4	5
Inertie	1138.4937	484.6838	378.3545	314.9732	267.5978

Table 1: Tableau de la distance totale en fonction du nombre de classes.

Pour K=2 classes, nous obtenons 2 groupes de 198 et 202 éléments.

2.1.2 Synth2 :

Comme pour Synth1, nous appliquons la méthode du coude sur les graphiques représentant la perte d'inertie en fonction du nombre de classes après classification à l'aide de l'algorithme des kmeans, ou celui avec distance adaptative.

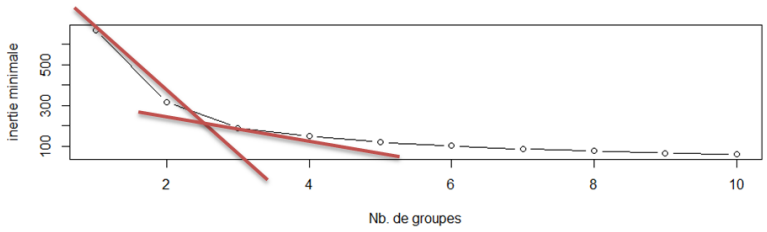

Figure 20: Perte d'inertie en fonction du nombre de classes (kmeans).

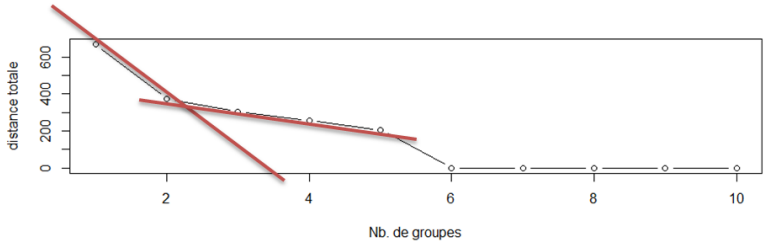

Figure 21: Perte d'inertie en fonction du nombre de classes (kmeans avec distance adaptative).

Nous voyons que avec l'algorithme des kmeans, 3 classes est ce qui parait le mieux, tandis que avec les kmeans avec distance adaptative, 2 classes est ce qui parait le mieux (en adéquation avec la classification réelle). Ces classes sont de 94 et 106 éléments, ce qui en fait des tailles proches de celles des classes réelles. En appliquant ajustedRandIndex, on obtient un indice élevé de 0.8830142, tandis que avec l'algorithme des kmeans classique cet indice est de 0.8456246 et on a des classes de 102 et 98 éléments.
Encore une fois, les deux algorithmes donnent des résultats différents, notamment lors du choix du nombre de classes, mais la classification finale avec le bon nombre de classes est toujours proche de la classification réelle.

2.1.3 Synth3 :

Enfin, nous faisons de même avec les données Synth3.

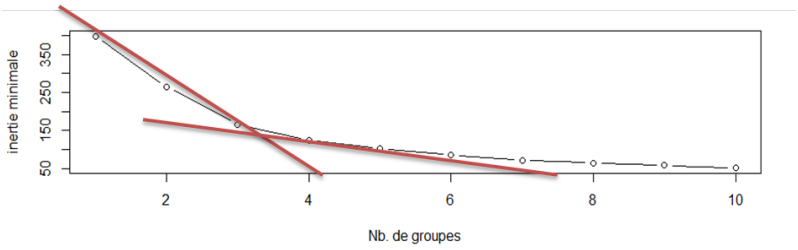

Figure 22: Perte d'inertie en fonction du nombre de classes (kmeans)

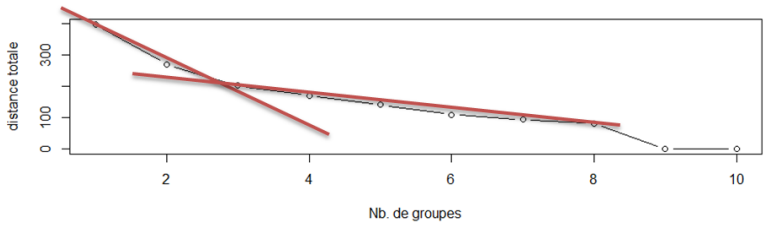

Figure 23: perte de distance en fonction du nombre de classes (kmeans Adaptative)

Cette fois ci, nous remarquons qu'il apparait avec la méthode du coude qu'il faut 3 classes dans le cas de l'algorithme des kmeans, et de même dans le cas de l'algorithme des kmeans avec distance adaptative, tandis que dans la réalité il n'y a que 2 classes.
Toutefois, nous pouvons effectuer une classification en 2 classes avec chaque méthode, et comparer l'adéquation avec la classification réelle. Nous comparons avec adjustedRandIndex.

- 2 classes avec kmeans : 0.1328602 et 2 classes de 73 et 127 éléments
- 2 classes avec kmeans distance adaptative : 0.5602924 et 2 classes de 103 et 97 éléments
- 3 classes avec kmeans : 0.437709 et 3 classes de 94, 45 et 61 éléments
- 3 classes avec kmeans distance adaptative : 0.4304443 et 3 classes de 94, 53 et 53 éléments.

Dans ce dernier cas, d'une part le choix du nombre de classes en se basant sur la méthode du coude n'est pas approprié dans les 2 cas si on veut la même classification que la classification réelle, mais d'autre part lorsque qu'on procède à la classification avec le bon nombre de classes (2), les résultats sont très différents de la réalité. Les deux algorithmes ne sont donc pas du tout adaptés à ce cas. Il y a cependant une meilleure adéquation avec l'algorithme des kmeans avec distance adaptative.

2.1.4 Conclusion

Nous pouvons donc conclure que selon les cas, l'algorithme des kmeans peut être très adapté (cas de Synth1), ou pas du tout (cas de Synth3). Il en est de même pour l'algorithme des kmeans avec distance adaptative. Nous remarquons tout de même que dans les 3 cas, c'était l'algorithme des kmeans avec distance adaptative qui était le plus adapté à la situation. Ceci est dû à l'utilisation de la distance de Mahalanobis plutôt que la distance euclidienne, on tient donc compte de la dispersion des points. Cependant, ces deux algorithmes ne sont parfois ni l'un ni l'autre adaptés, et il faut donc réfléchir à un nouvel algorithme dérivé des kmeans, peut être par choix d'une autre méthode de calcul des distances.

2.2 Données réelles

2.2.1 Iris

Les données iris ont 150 observations de 4 variables quantitatives, et une variable qualitative z représentant la classe. Il y a 3 classes de 50 éléments chacun.

Avec l'algorithme des kmeans :

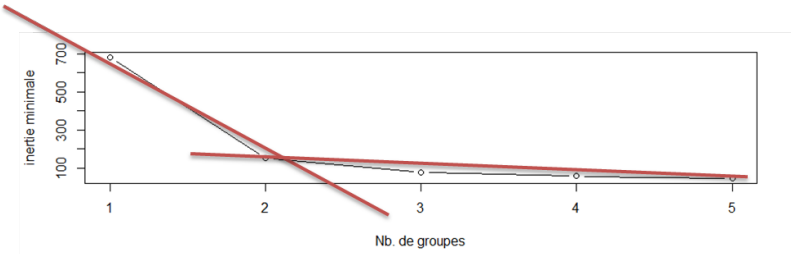

Figure 24: Graphe montrant la perte d'inertie en fonction du nombre de classes (kmeans)

Avec notre algorithme :

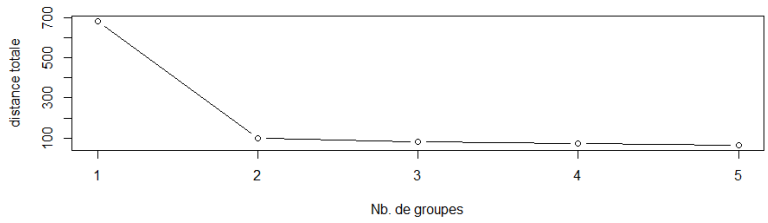

Figure 25: Graphique montrant la perte de distance totale en fonction du nombre de classes

K	1	2	3	4	5
Inertie	681.37060	152.34795	78.85144	57.22847	46.44618
Distance Totale	681.37060	98.64716	82.85021	71.30603	65.08072

Table 2: Tableau montrant l'inertie et la distance totale en fonction du nombre de classes k

Nous voyons que pour K=1, le critère optimisé est le même quelle que soit la méthode utilisée. Cependant, les valeurs des critères pour k=2 à 5 sont différentes selon l'algorithme utilisé. Cela est dû au fait que pour K=1, l'inertie est égale à la distance totale. Dans le cas des kmeans classique, la meilleure partition semble être avec k=3 tandis que avec l'algorithme des kmeans adapté celle-ci semble être meilleure pour k=2.

Pour K=2 : Kmeans :

- Inertie : 152.348

- 2 classes de 97 et 53 éléments
- AdjustedRandIndex : 0.5399218

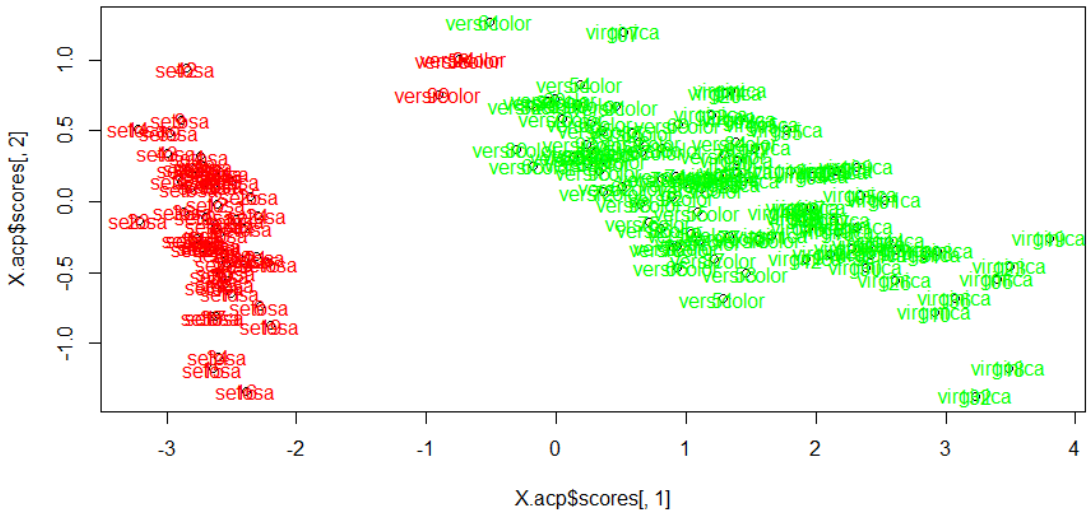

Figure 26: Classification des éléments pour K=2 et avec l'algorithme des Kmeans

Kmeans Ada :

- Distance Totale: 100.7116
- 2 classes de 87 et 63 éléments
- AdjustedRandIndex : 0.2167627

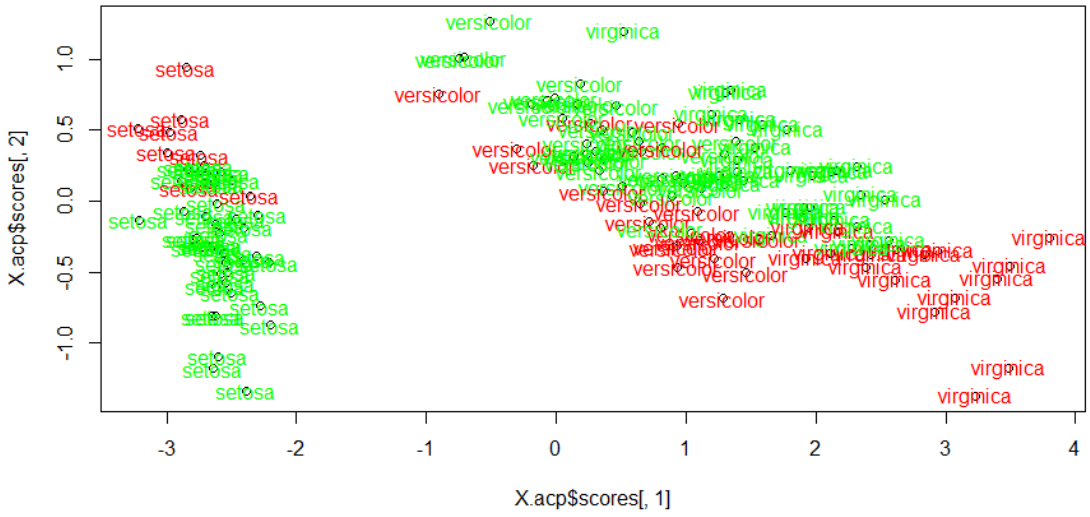

Figure 27: Classification des éléments pour K=2 et avec l'algorithme des Kmeans avec distance adaptative

Pour K=3:

Kmeans :

- Inertie : 142.7535
- 3 classes de 38, 62, 50 éléments
- AdjustedRandIndex : 0.7302383

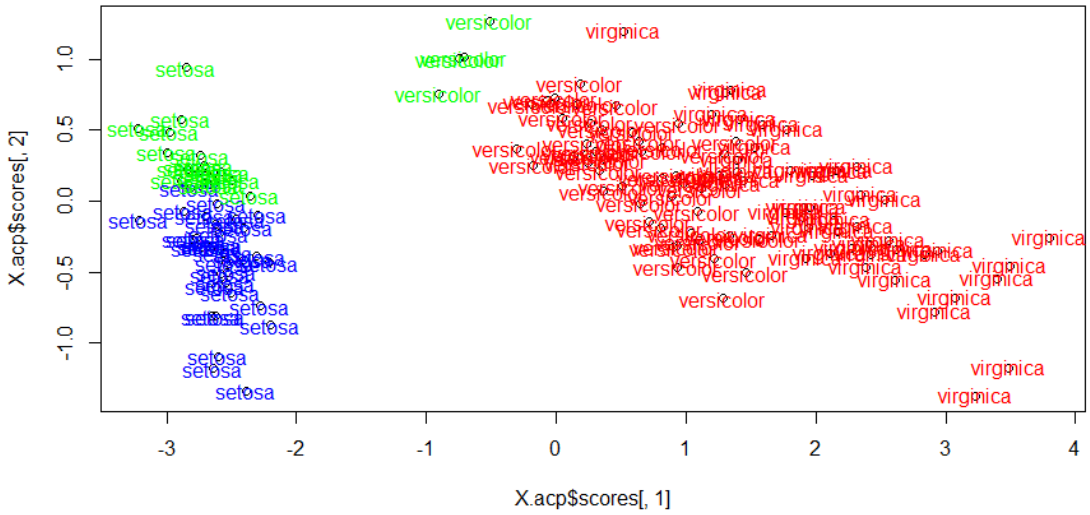

Figure 28: Classification des éléments pour K=3 et avec l'algorithme des Kmeans

Kmeans Ada :

- Distance Totale: 81.3536
- 3 groupes de 48, 51 et 51 éléments
- AdjustedRandIndex : 0.05127749

Nous remarquons que plus K augmente, plus les groupes sont de taille similaire avec l'algorithme des Kmeans avec distance adaptative. Nous pouvons supposer que celui-ci devient de plus en plus adapté lorsque la classification doit être précise (avec k qui augmente). Cependant, avec adjustedRandIndex nous observons que les classifications obtenues ne sont que très peu cohérentes avec la classification réelle. C'est ce qui apparaît aussi lorsque nous représentons les données sur un graphe.

2.2.2 Spam

Le jeu de données Spam comporte 4601 observations de 58 variables (57 quantitatives, et une qualitative z donnant la classe réelle). Ces données sont séparées dans 2 classes de respectivement

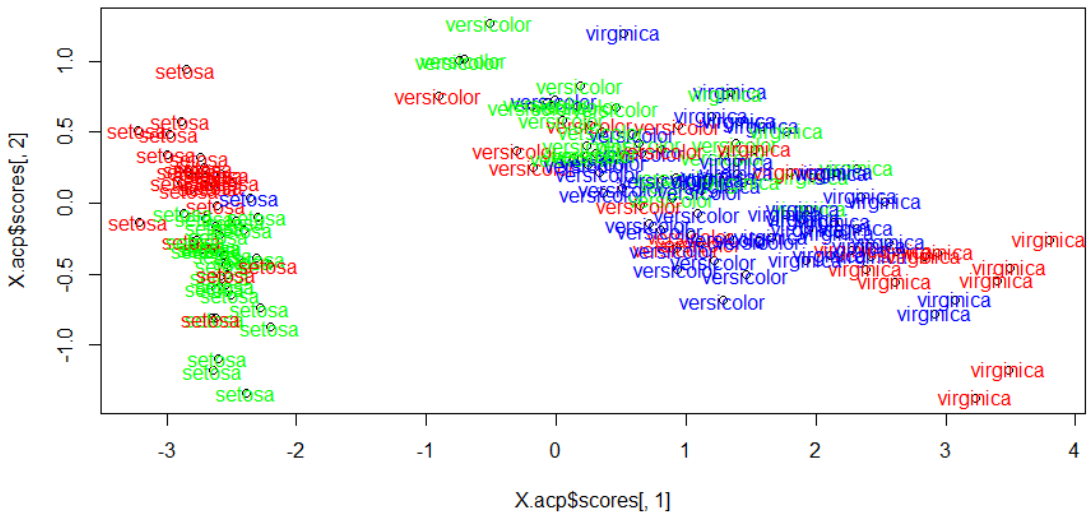

Figure 29: Classification des éléments pour K=3 et avec l'algorithme des Kmeans avec distance adaptative

1813 et 2788 éléments.
En effectuant une ACP et en l'affichant sur un graphe, nous voyons qu'il y a des données très dispersées, du bruit. Les deux classes sont ici représentées de couleur différente.

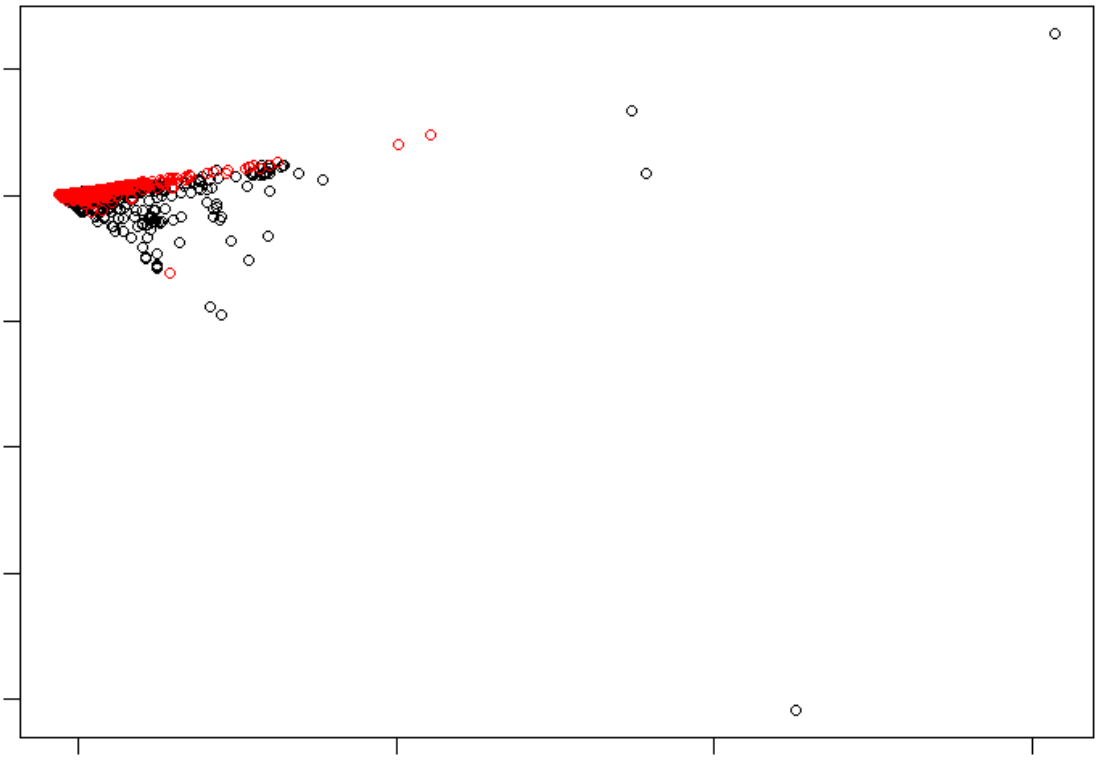

Figure 30: Graphe de l'ACP sur les données Spam.

1ère approche

Ces données « aberrantes » risquent de fausser le calcul des centres de gravités dans les algorithmes des kmeans. Une bonne solution serait donc de s'en séparer et de n'effectuer la classification que sur les données « proches » et jugées utiles.
Nous décidons de supprimer les valeurs considérées comme aberrantes sur les boxplots, à l'aide de la fonction boxplot.stats. Nous appliquons cette fonction sur l'ACP des données, en ne gardant que les deux premières colonnes, afin de ne pas supprimer trop de données inutilement. En effet, si on supprime toutes les données ayant une valeur aberrante dans une des 57 colonnes, alors nous nous retrouvons avec seulement 189 éléments. En appliquant sur la sous matrice ayant les deux premières colonnes de x.acp afin d'avoir le maximum d'information, nous nous retrouvons avec 3497 éléments.

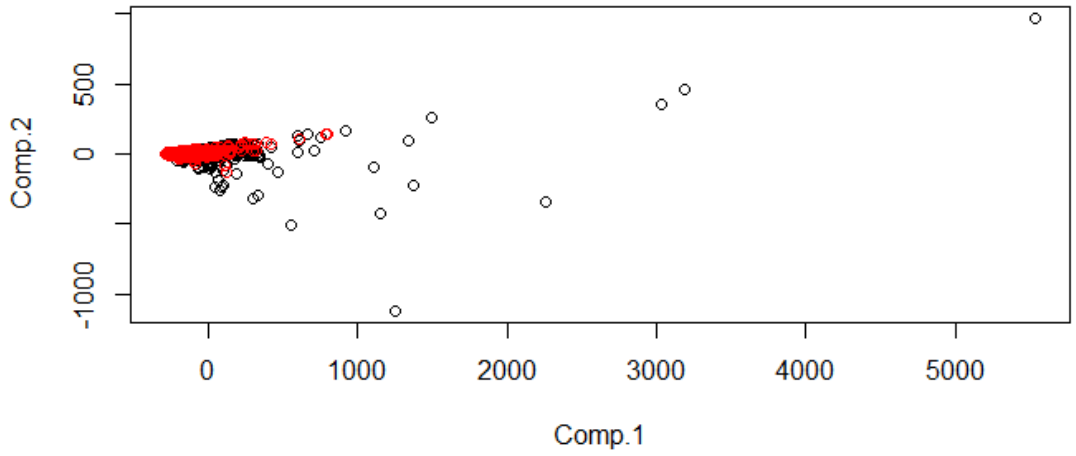

Figure 31: Graphe de l'ACP sur les données SPAM après élimination de valeurs aberrantes.

Nous voyons qu'il reste tout de même des valeurs assez éloignées, mais de nombreuses ont déjà été éliminées. Nous décidons de ne pas en supprimer plus afin de garder de l'information. Nous appliquons ensuite les deux algorithmes des kmeans sur ce nouveau set de données.

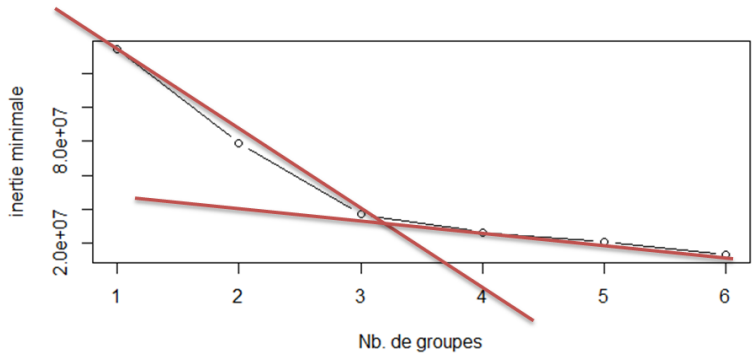

Figure 32: Graphique de la perte d'inertie en fonction du nombre de classes (kmeans).

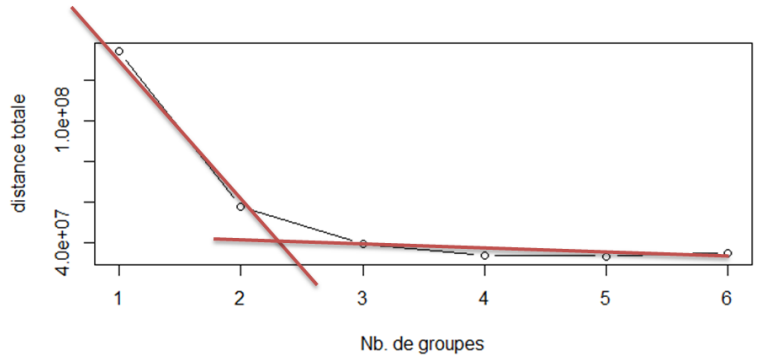

Figure 33: Graphique de la perte de distance totale en fonction du nombre de classes (kmeans adaptatifs).

Nous remarquons avec la méthode du coude que avec les deux méthodes, 3 classes semblent être nécessaire pour une bonne partition, même si le coude est placé plus proche de 2 pour les kmeans adaptatifs que le classique (et donc plus proche du nombre de classe réel qui est de 2).

Méthode	Kmeans	Kmeans adaptatif
AdjustedRandIndex	0.09073962	0.04625688

Table 3: Tableau montrant la similitude entre les classifications obtenues et la classification réelle

Les résultats obtenus sont malheureusement très peu proches de la classification réelle, quel que soit l'algorithme des Kmeans choisit. Nous nous interessons donc à une deuxième approche.

2ème approche

La méthode des K-means ne permet pas d'effectuer une classification correcte des données. Cela peut s'expliquer par plusieurs facteurs :

- Des données aberrantes qui faussent le calcul des centres dans le cas où on n'applique pas l'ACP
- Une perte d'informations trop élevée lorsqu'on applique l'ACP

Nous nous sommes donc tournés vers une autre méthode : les machines à vecteurs de support (SVM). Cette méthode n'a pas été vue en SY09 mais nous avons eu l'occasion de l'utiliser dans d'autres circonstances.
Les SVMs sont des techniques d'apprentissage supervisés destinées à résoudre des problèmes de discrimination. Pourquoi cela convient-il au problème ?

- Nous sommes face à un problème de classification
- Nous connaissons les « vraies » classes (spam/non-spam) : il est donc possible d'utiliser l'apprentissage supervisé
- Nous disposons de plusieurs milliers de données

Voilà la démarche suivie (utilisation des packages « caret » et « kernlab »):

1. Séparation des données La première étape fut de séparer les données Spam (de manière aléatoire) en 2 parties :

   - 80% des données sont stockées dans un dataSet « d'entrainement » *(dataTrain)*
   - 20% des données sont stockées dans un dataSet « de test » *(dataTest)*

   dataTrain est utilisé pour la phase d'apprentissage du modèle tandis que *dataTest* est utilisé pour tester le modèle. Il est essentiel d'effectuer cette séparation pour obtenir des résultats non-biaisés : En effet utiliser les mêmes données pour l'apprentissage et le test aurait tendance à augmenter la performance du modèle (plus de résultats exacts)

2. Phase d'apprentissage C'est ici qu'on utilise *dataTrain* pour créer le modèle, voilà les étapes que nous avons suivies :

   - Déterminer les bons paramètres pour le SVM
   - Entrainer le modèle

3. Test sur le *dataTest* Une fois le modèle entrainé, nous avons testé ce dernier sur notre dataTest puis fait une matrice de confusion entre les résultats annoncé par le modèle et les vraies données. Voilà les résultats obtenus :

Prediction/Reference	1	2
1	328	17
2	34	540

Table 4: Matrice de confusion.

On obtient une précision générale de **94%**, ce qui est parfaitement acceptable.

3 Justification

4 Annexes

4.1 Commandes R

4.1.1 Partie 1

```
#1
#only quant variables
recette.quant <- recette[,-1]
#affichage par origine
plot(recette[,2],recette[,3], col=recette$origin)
plot(recette[,2],recette[,5], col=recette$origin)
#moyenne du nombre d'ingredients par recette
mean(rowSums(recette.quant>0))
#poids moyen des recettes
mean(rowSums(recette[,-1]))

corRecette <- cor(recette.quant)

#count the nb of positive values for each row in a matrix
nbNot0 <- function(tab){
  nc <- ncol(tab);
  nr <- nrow(tab);
  l <- vector("list",0);
  for(j in 1:nr){
    n <- 0;
    for(i in 1:nc){
      if(tab[j,i]>0)
        n<-n+1;
    }
    l <- c(l,n);
  }
  return(l);
}

l<-nbNot0(recette.quant)
mean(unlist(l))


#2
#prcomp car plus de variables que d'observations
recette.acp <- prcomp(recette.quant)
plot(recette.acp$x, col=recette$origin)
summary(recette.acp)
#importance de chaque composante principale
plot(recette.acp)


#3 - CAH
#matrice de distance
rownames(recette.quant) <- recette$origin
recette.cah <- dist(recette.quant, method="manhattan")
recette.cahWard <- hclust(recette.cah, method="ward.D2")
plot(recette.cahWard)
#dendrogramme avec materialisation des groupes
rect.hclust(recette.cahWard,k=3)

recette.sh <- cmdscale(recette.cah, eig = TRUE)
plot(recette.sh$points, main = "AFTD")
```

```
text(recette.sh$points[,1],recette.sh$points[,2], recette$origin)
shepard <- Shepard(recette.cah, recette.sh$points)
plot(shepard)


#4 - par kmeans
recette.cr <- scale(recette.quant, center=T, scale=F)
recette.kmeans <- kmeans(recette.cr,centers=3,nstart=10)
inertie.minmut <- rep(0,times=10)
for (k in 1:10){ clus <- kmeans(recette.cr,centers=k,nstart=100)
inertie.minmut[k] <- clus$tot.withinss }
plot(1:10,inertie.minmut,type="b",xlab="Nb. de groupes",ylab="inertie minimale")
recette.kmeans$cluster

#5 - classification geographique
recette.groupes.cah <- cutree(recette.cahWard,k=3)
plot(recette.acp$x[,1],recette.acp$x[,2])
text(recette.acp$x[,1],recette.acp$x[,2],col=c("red","green","blue")[recette.groupes.c

#test avec 4 groupes
recette.groupes.cah4 <- cutree(recette.cahWard,k=4)
plot(recette.acp$x[,1],recette.acp$x[,2])
text(recette.acp$x[,1],recette.acp$x[,2],col=c("red","green","blue","yellow")[recette


#6 - recettes_echeant
rec.ech <- read.csv("donnees/donnees/recettes-echant.txt")
corRecEch <- cor(rec.ech[,-1])
#acp
rec.ech.acp <- princomp(rec.ech[,-1])
plot(rec.ech.acp)
rec.ech.acp$loadings

# 7
install.packages(pkgs="plyr")
library(plyr)
#fait la moyenne de chaque colonne
rec.ech.trans <- ddply(rec.ech, .(origin), colwise(mean))
rownames(rec.ech.trans) <- rec.ech.trans$origin
#on enleve l'origine et on fait la transposee pour avoir une rotation du tableau
rec.ech.trans.transpo <- t(rec.ech.trans[,-1])
rec.ech.dist <- dist(rec.ech.trans.transpo, method="manhattan")
rec.ech.trans.ACP <- princomp(rec.ech.dist)
plot(rec.ech.trans.ACP$scores[,1],rec.ech.trans.ACP$scores[,2],col=rec.ech$origin)

#kmeans pour determiner nb de classes
recette.ech.cr <- scale(rec.ech[,-1], center=T, scale=F)
inertie.ech.minmut <- rep(0,times=8)
for (k in 1:10){ clus <- kmeans(recette.ech.cr,centers=k,nstart=100)
inertie.ech.minmut[k] <- clus$tot.withinss }
plot(1:10,inertie.ech.minmut,type="b",xlab="Nb. de groupes",ylab="inertie minimale")

#8
rec.ech.cahWard <- hclust(rec.ech.dist, method="ward.D2")
plot(rec.ech.cahWard)
#4 classes semblent se distinguer
rect.hclust(rec.ech.cahWard,k=4)
rec.groupes.cah4 <- cutree(rec.ech.cahWard,k=4)
```

```
plot(rec.ech.trans.ACP$scores[,1],rec.ech.trans.ACP$scores[,2])
text(rec.ech.trans.ACP$scores[,1],rec.ech.trans.ACP$scores[,2],col=c("red","green","b
```

```
#9
library(cluster)
rec.ech.K <- pam(rec.ech.dist, 4)
rec.ech.K$medoids
```

4.1.2 Partie 2

References