Université de Technologie de Compiègne
SY09 : Analyse des données et Data mining

Compte-Rendu Projet1 : Application les méthodes automatiques

Jingyi Hu & Bastien ROLANDO

Printemps 2018

1 Cuisine

Cette partie concerne un jeu de données sur les recettes de cuisine des différents pays. L'objectif de cette étude est de pouvoir classifier des recettes en fonction de leur ingrédients. L'ensemble du code utilisé pour la démarche est disponible dans le fichier *cuisine.R*.

1.1 Analyse exploratoire du jeu de données

Le jeu de données est un data frame qui contient 26 lignes et 50 colonnes. Chaque ligne correspond à un pays, chaque colonne correspond à un ingrédient. Pour chaque pays, on a le pourcentage d'utilisation de chaque ingrédient dans un ensemble représentatif de recettes de cuisine.

1.2 Analyse en ACP du jeu de données

Nous réalisons une ACP avec la fonction *prcomp()* de R et non pas *princomp()* car celle-ci ne fonctionne pas si on a plus de colonnes(attributs) que de lignes(enregistrements), ce qui est le cas ici.

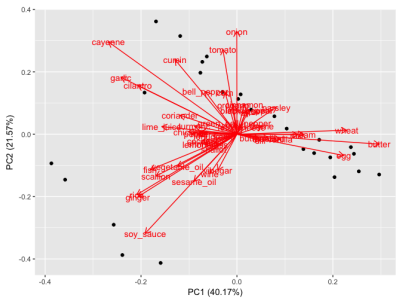

Figure 1: Plot avec les 2 composantes principales

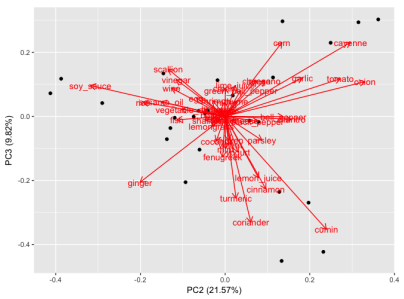

Figure 2: Plot avec les composantes principales2,3

Sur le premier graphique (Figure 2), nous étudions seulement les deux premières composantes, nous nous rendons compte qu'il y a énormément d'attributs, et qu'il est par conséquent compliqué d'avoir un graphique lisible. Sur le second graphique (Figure 1), nous étudions seulement les composantes 2 et 3 pour faire la comparaison entre les deux premières composantes.

1.3 Analyse ascendante hiérarchique du jeu de données

En premier lieu, nous calculons les distances de Manhattan du jeu de données, ensuite nous réalisons une *heatmap* afin d'avoir une vision globale du jeu de données et de définir un nombre de classe approximatif. Voir figure 3.
Dans notre heatmap, nous avons 3 couleurs principales : Le jaune, l'orange et le rouge, par conséquent il semble sage de choisir k=3 classes pour notre classification.
Quant à la réalisation du dendrogramme (Figure 4),nous avons essayé plusieurs méthodes avec la fonction *hclust()* dans R avec différentes critères d'agrégation: min, max, moyenne, moyenne pondérée, Distance euclidienne, ward, puisque les résultats sont ressemble,nous ne montrons ici que celui avec la critère d'agrégation min. Nous observons que notre classification n'est pas dénuée de sens : en effet, le premier cluster contient des pays tels que le Japon, la Chine ou le Vietnam, c'est à dire des cuisines de type asiatique. Le second cluster contient des pays comme les États-unis, la France, le Royaume-Uni ou encore l'Allemagne, c'est à dire une cuisine d'influence européenne. Et enfin le 3ème cluster semble désigner une famille d'influence latine, puisqu'elle contient des pays tels que l'Italie, la Grèce, et l'Amérique du Sud. Il y a toutefois l'Inde qui est incluse dans ce cluster, cela ne semble pas tout à fait adéquat. On observe tout de même que sa distance par rapport aux autres membres de ce cluster est parfois élevée.

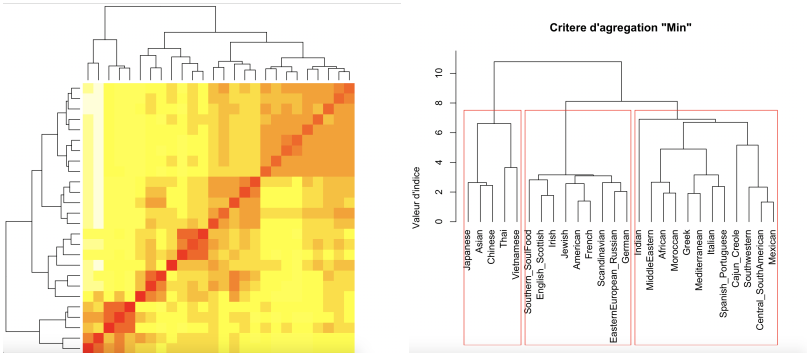

Figure 3: Heatmap du jeu de données

Figure 4: Dendrogramme avec k = 3

1.4 Groupement de données avec l'algorithme des K-Means

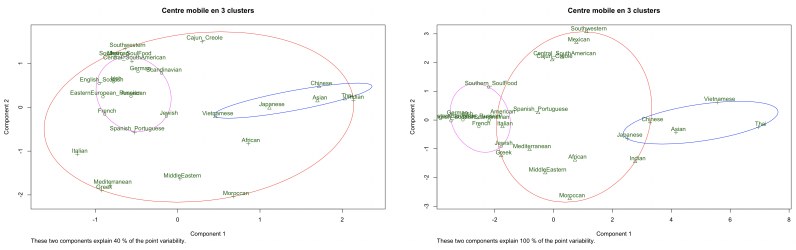

Figure 5: Kmeans avec k = 3 et composantes 1 - 2

Figure 6: Kmeans avec k = 3 et composantes 1 - 3

En nous basant toujours sur les distances, nous réalisons une classification avec la méthode des K-Means, en se basant toujours sur la distance de Manhattan (Figure 5 et 6).

Le premier graphique est réalisé à partir des deux premières composantes, celles-ci expliquent très bien les clusters européens et asiatiques, mais elles restent peu adaptées pour les autres pays. Nous utilisons donc la 1ère composante et la 3ème dans un second graphique, et nous obtenons une classification mieux ordonnée pour les 3 clusters, mais avec une distance intra-cluster moyenne plus élevée.

1.5 Classification géographique des origines

Dans la figure 5, on retrouve nos 3 classes, et on remarque que le 3ème cluster, qui englobait à la fois les cuisines méditerranéennes, sud-américaines et indiennes, est très général et a une distance intra-cluster très élevée. Cela explique pourquoi tous ces pays ont été inclus dans le même cluster. D'un autre côté, on observe que les cuisines asiatiques et européennes forment en effet un cluster avec des distances intra-cluster très faibles.

1.6 Analyse descriptive du jeu complet de données

Les données étudiées précédemment proviennent d'une simplification d'un data frame contenant un ensemble de recettes de différents pays/de régions. Nous allons étudier ce data frame par la suite.
Celui-ci contient 2000 lignes et 50 colonnes. Les colonnes sont les mêmes que les précédentes, c'est à dire des ingrédients. Chaque ligne du data frame représente une recette d'un pays. Pour chaque recette, une valeur booléenne représente la présence de chaque ingrédient dans la recette (0 si absent, 1 si présent).

1.7 Dissimilarité entre les ingrédients

Pour obtenir un tableau individu/variable portant sur les ingrédients, on peut travailler à partir de la transposée de la matrice étudiée dans les premières questions. En effet, on obtient ainsi pour chaque ingrédient son utilisation dans les pays. Ensuite, on peut travailler à partir de la matrice de covariance pour étudier la similarité entre les différents ingrédients.

1.8 Classification ascendante hiérarchique de la dissimilarité

Comme précédemment, on observe le heatmap afin de déterminer un nombre approximatif de classes, on en observe entre 5 et 6 couleurs dans celui-ci :

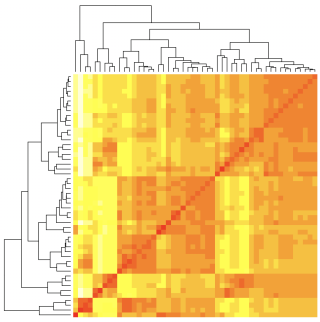

Figure 7: Heatmap sur les ingrédients

On affiche donc la classification ascendante hiérarchique avec un découpage en 5 classes et on obtient le dendrogramme suivant

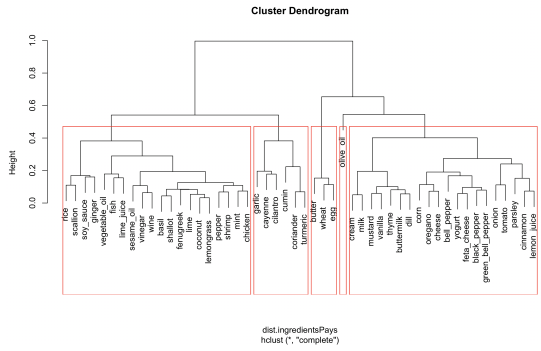

Figure 8: Dendrogramme avec 5 classes sur les ingrédients

On observe que certains groupes ont du sens, tel que le cluster blé, oeuf et beurre, qui semble être un cluster qui comprend les recette de patisserie. Nous avons un autre cluster qui représente les herbes et épices. Pour le reste, c'est assez peu significatif. Il y a également un cluster avec uniquement l'huile d'olive, ce qui est assez étrange.

1.9 Ingrédients représentants de chaque classe

Ici, on utilise la méthode des k-medoides afin de constituer les clusters. Cette méthode minimise la distance absolue entre le centre et les points alors que la méthode des k-means minimise la distance au carré. Cela à pour conséquence d'être moins sensible aux extrêmes et au bruit, et par conséquent il s'agit d'une méthode plus robuste.

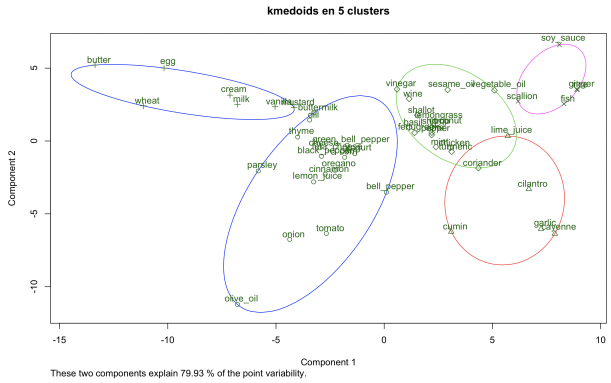

Figure 9: Clusters avec méthode des k-medoids à 5 classes

2 Classification par K-means avec distance adaptative

2.1 Algorithme des K-means avec distance adaptative

L'algorithme développé suit les recommandations et les pistes de l'énoncé, chaque morceau est détaillé autant que possible avec des commentaires directement dans R dans le fichier *kmeans_ adp.R* .

2.2 Test de l'algorithme sur des données synthétiques

Nous testons l'algorithme sur différents ensembles de données qui n'ont pas réellement d'interprétation possible afin de s'assurer qu'il fonctionne. Dans chaque cas, nous allons comparer le résultat de notre algorithme avec celui de la fonction *kmeans* de R. L'ensemble du code concernant les données synthétiques est situé dans le fichier *synth.R*.

2.2.1 Premier jeu de données synthétiques

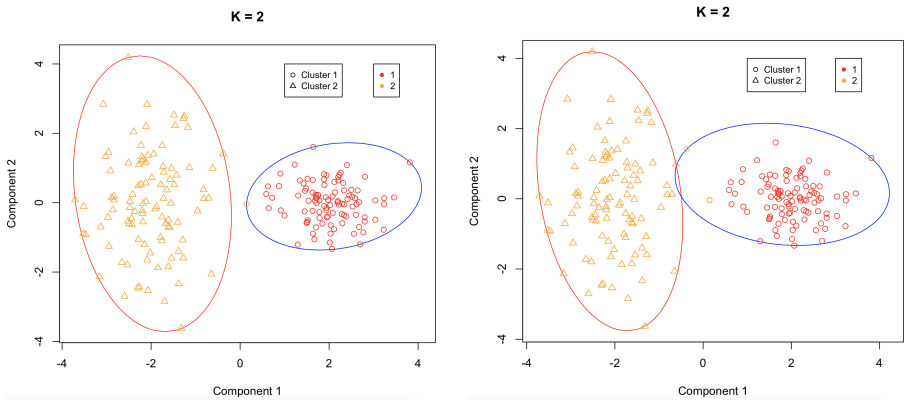

Figure 10: Plot avec utilisation de la fonction *kmeans*

Figure 11: Plot avec utilisation de l'algorithme en 2.1

De première vue, les deux classifications sont très similaires, les données se prêtent bien à ce genre de traitement puisqu'on arrive facilement à distinguer deux classes.

- La classification réalisée avec la fonction *kmeans* a une adéquation de 0.98, donc presque aucune erreur.
- La classification réalisée avec la fonction l'algorithme des k-means a distance adaptative a une adéquation de 0.96, on a encore un résultat presque parfait, mais légèrement inférieur à celui de la fonction *kmeans*.

2.2.2 Second jeu de données synthétiques

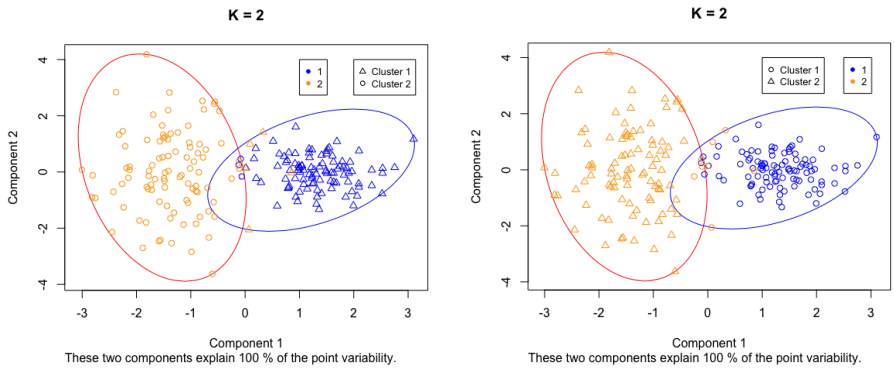

Figure 12: Plot avec utilisation de la fonction *kmeans*

Figure 13: Plot avec utilisation de l'algorithme en 2.1

Ici encore, les données se prêtent bien à une classification même si elles se croisent un peu. De même, on obtient des classifications assez similaires.

- Les deux classifications ont une adéquation de 0.85, donc quelques erreurs. On observe qu'il n'y a pas de méthode meilleure que les autres dans ce cas. *kmeans*

2.2.3 Troisième jeu de données synthétiques

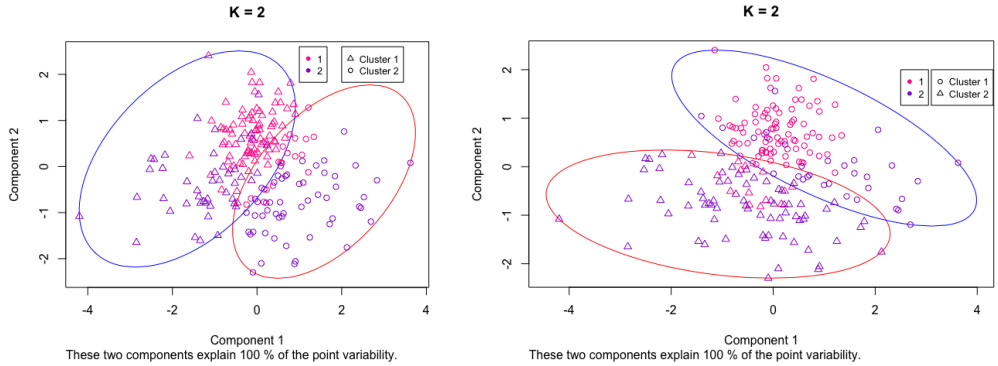

Figure 14: Plot avec utilisation de la fonction *kmeans*

Figure 15: Plot avec utilisation de l'algorithme en 2.1

Dans ce cas ci, les données sont extrêmement entrecroisées. Il est dur de déterminer des classes d'un seul coup d'œil. Les deux classifications sont également très différentes.

- La classification réalisée avec la fonction *kmeans* a une adéquation de 0.044, donc presque totalement fausse.
- La classification réalisée avec la fonction l'algorithme des k-means a distance adaptative a une adéquation de 0.29, ce qui est nettement meilleur que la fonction *kmeans*, même si ce résultat n'est pas très bon.

2.2.4 Conclusion sur les exemples synthétiques

On observe grâce aux cas précédents que notre algorithme des k-means a distance adaptative a été autant efficace dans les cas "simples", et beaucoup plus efficace dans les cas complexes ou les données sont entrecroisées ou sous forme d'ellipse.

2.3 Test de l'algorithme sur des données réelles

Nous avons vérifié l'efficacité de notre algorithme des k-means à distance adaptative, il est maintenant temps de l'utiliser sur des exemples concrets.

2.3.1 Données iris

L'ensemble du code concernant les données iris est situé dans le fichier *iris.R*. Nous allons tenter de classifier les données iris avec la méthode *kmeans* de R et avec notre méthode des kmeans à distance adaptative. Tout d'abord, nous allons chercher à définir le nombre de classes optimal pour les iris à partir de l'inertie intra-classe minimale. Pour cela, nous la calculons pour K = 2,3,...,10.

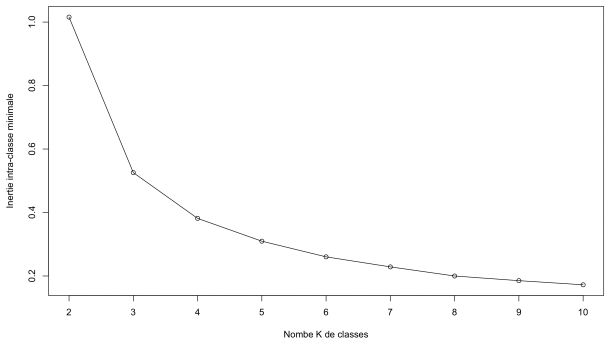

Figure 16: Inertie intra-classe en fonction du nombre de classes

On observe que le saut se situe entre 2 et 3 classes, par conséquent, le nombre de classes optimal est situé entre 2 et 3. Nous allons donc commencer par faire une partition des données pour K=2,3 avec la méthode *kmeans* de R. Puis la même chose avec la méthode des kmeans à distance adaptative.

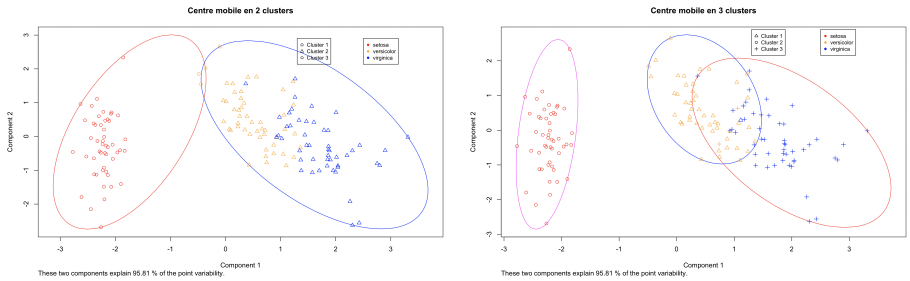

Figure 17: Clusters avec la méthode *kmeans* de R pour K = 2

Figure 18: Clusters avec la méthode *kmeans* de R pour K = 3

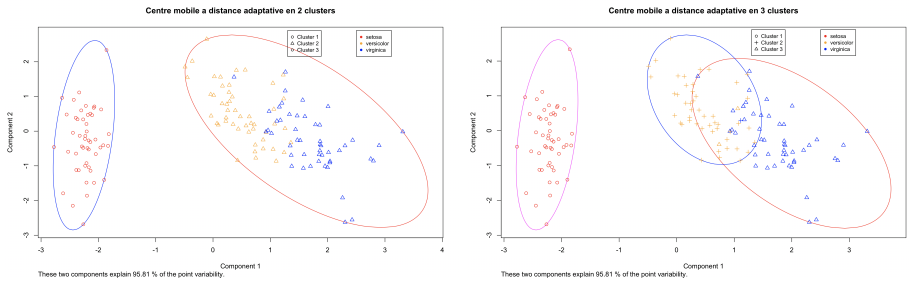

Figure 19: Clusters avec la méthode des kmeans adaptatifs pour K = 2

Figure 20: Clusters avec la méthode des kmeans adaptatifs pour K = 3

On observe un partitionnement assez proche dans les 2 cas, la majeure différence se situe sur le cluster de gauche, il s'agit de l'espèce *setosa*. Dans la méthode des *kmeans* a 2 clusters, le partitionnement est assez mal défini et on dispose d'une distance intra-cluster assez élevée, toutefois lorsqu'on passe a 3 classes, la partition est bien définie et quasiment identique à celle des kmeans adaptatifs.

2.3.2 Données spam

L'ensemble du code concernant les données spam est situé dans le fichier *spam.R*. Dans les données suivantes, chaque ligne représente un email. Pour chaque email, la présence de certain mots est

donnée en pourcentage. Nous disposons également de la longueur de l'email.

Après analyse exploratoire du jeu de données, on observe qu'il s'agit d'une matrice creuse, et que les quantités présentes sont globalement très petites. Par conséquent, on va simplifier le jeu de données en s'intéressant uniquement à la présence ou non d'un mot dans un mail.

Pour cela, on supprime les dernières colonnes concernant la longueur, et on garde uniquement celles qui concernent les mots. Ensuite, on transforme la matrice en matrice binaire : Si un mot est présent dans un mail (c'est à dire que la quantité dans la colonne est supérieure à 0), la valeur de la colonne vaut 1, sinon elle reste a 0.

Pour transformer des variables corrélées en nouvelles variables non-corrélées les unes aux autres et ainsi réduire le nombre de variables pour de rendre l'information moins redondante, on a besoin d'utiliser une Analyse en composantes principales(ACP) dans notre jeu de données. Avant de faire cela, on a appliqué la fonction *scale()* dans R. On voit que la dernière partition à apporté un gain informationnel significatif avec k=2 classes.

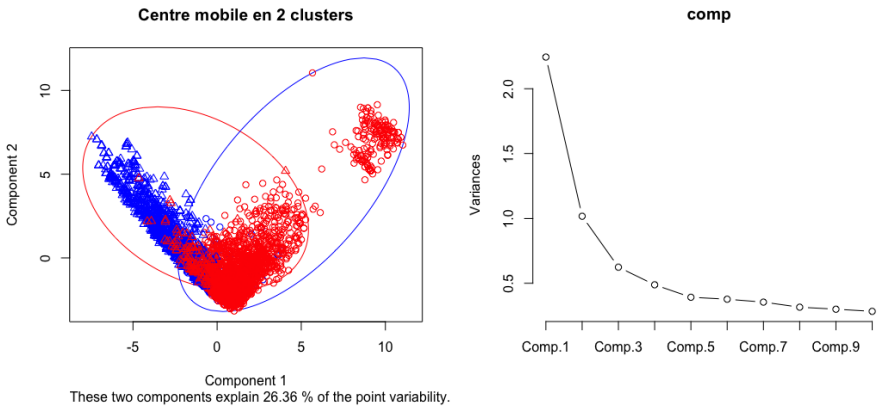

Figure 21: Plot avec les 2 composantes principales

Figure 22: inertie intra-classe en fonction du nombre de classes

Pour la suite, on a appliqué les *kmeans* de R et les kmeans à distance adaptative sur le jeu de données originel et les deux premières composantes. On a utilisé la fonction *adjustedRandIndex()* pour évaluer les différents résultats.

	adjustedRandIndex
K-means	0.498512
ACP + Kmeans	0.4773322
pam	0.2563789
ACP + pam	0.4737757
ACP + Kmeans adaptifs sur composante 1 et 2	0.3941267
ACP + Kmeans adaptifs sur composante 1 et 4	0.5318719

On observe que d'une manière générale, effectuer une ACP en amont permet d'obtenir de meilleurs résultats puisqu'on arrive a trouver des composante assez efficace pour étudier le jeu de données. Ensuite, le meilleur résultat obtenu est en utilisant la 1ère et la 4ème composante après ACP et avec les kmeans à distance adaptative, ainsi on obtient une adéquation de presque 0.54. Cela s'explique sans doute par le fait que dans ces deux composantes, les données sont moins mélangées et mieux réparties sous forme d'ellipse, ce qui fonctionne assez bien avec cette méthode.

Par soucis de comparaison, nous avons également utilisé des méthodes supervisées sur le jeu de données originel, voici les résultats :

	taux d'erreur
QDA	0.1679
QDA + ACP	0.1694
LDA	0.1149
LDA + ACP	0.1128
Random Forest	0.0894
Random Forest + ACP	0.1099
Logistic regression	0.072

On observe que les méthodes supervisées sont plus adaptées à ce genre de jeu de données, par exemple avec une régression logistique, nous n'avons que 0.072 de taux d'erreur. Le taux d'erreur le plus élevé est de seulement 0.1679, ce qui reste bien en dessous des taux obtenus avec les méthodes non-supervisées.

2.4 Justification

2.4.1

$$J(\{v_k, M_k\}) = \sum_{k=1}^{K} \sum_{i=1}^{n} z_{ik} d_{ik}^2 = \sum_{k=1}^{K} \sum_{i=1}^{n} z_{ik} (x_i - v_k)^T M_k (x_i - v_k)$$

On suppose que x_i^e est la e ème élement de x_i:

$$|| \ x_i \text{ - } v_k \ ||_2^2 = \sum_{e=1}^{p} (x_i^e - v_k^e)^2$$

$$\frac{\partial L}{\partial v_k} = \begin{pmatrix} \frac{\partial L}{\partial v_k^1} \\ \frac{\partial L}{\partial v_k^2} \\ \dots \\ \frac{\partial L}{\partial v_k^p} \end{pmatrix}$$

$$J = \sum_{i,k,e} z_{i,k} Mk (x_i^e - v_k^e)$$

À chaque itération, pour chaque classe k:

$$\frac{\partial L(\{v_k, M_k, \lambda_k\})}{\partial v_k^e} = \sum_{i=1}^{n} z_{ik} M_k 2(-1)(x_i^e - v_k^e)$$

On calcule v_k:

$$\frac{\partial L(\{v_k, M_k, \lambda_k\})}{\partial v_k} = \sum_{i=1}^{n} z_{ik} M_k (-2)(x_i - v_k)$$

$$\sum_{i=1}^{n} z_{ik} M_k (-2)(x_i - v_k) = 0$$

$$\sum_{i=1}^{n} z_{ik} (x_i - v_k) = 0$$

$$\sum_{i=1}^{n} z_{ik} x_i - n_k v_k = 0$$

$$v_k = \frac{1}{n_k} \sum_{i=1}^{n} z_{ik} x_i$$

Alors on a montré que la mise à jour des prototypes revient à calculer les centres de gravité des classes.

2.4.2

a) Si on prend la formule dans le document:

$$\frac{\partial L(\{v_k,M_k,\lambda_k\})}{\partial M_k} = \sum_{i=1}^{n} z_{ik}(x_i - v_k)(x_i - v_k)^T - \lambda_k det(M_k)(M_k^{-1})^T$$

Si on met $\frac{\partial L}{\partial M_k} = 0, alors$:

$$\sum_{i=1}^{n} z_{ik}(x_i - v_k)(x_i - v_k)^T - \lambda_k det(M_k)(M_k^{-1})^T = 0$$

$$(M_k^{-1})^T = \sum_{i=1}^{n} z_{ik}(x_i - v_k)(x_i - v_k)^T \cdot \lambda_{-1} \cdot \rho_k^{-1}$$

$$V_k = \frac{1}{n_k} \sum_{i=1}^{n} z_{ik}(x_i - v_k)(x_i - v_k)^T - \lambda_k det(M_k)(M_k^{-1})^T$$

$$(M_k^{-1})^T = \rho_k^{-1} \cdot \lambda_k^{-1} \cdot n_k \cdot V_k$$

On a V_k est une matrice symétrique:

$$M_k^{-1} = \rho_k^{-1} \cdot \lambda_k^{-1} \cdot n_k \cdot V_k \quad (1)$$

Ainsi det(M) = ρ_k :

$$det(M^{-1}) = \rho_k^{-1}$$

$$det(\rho_k^{-1} \cdot \lambda_k^{-1} \cdot n_k \cdot V_k) = \rho_k^{-1}$$

Comme la dimension de V_k est p × p, on a:

$$(\rho_k^{-1} \cdot \lambda_k^{-1} \cdot n_k)^p \cdot det(V_k) = \rho_k^{-1}$$

$$(\rho_k^{-1} \cdot \lambda_k^{-1} \cdot n_k)^p = (det(V_k))^{-1} \cdot \rho_k^{-1}$$

$$\rho_k^{-1} \cdot \lambda_k^{-1} \cdot n_k = (\rho_k det(V_k))^{-1/p} \quad (2)$$

On met l'équation (2) dans l'équation (1), alors on a:

$$M_k^{-1} = (\rho_k det(V_k))^{-1/p} \cdot V_k$$

On a alors l'équation (4) dans l'énoncé.

b)

$$\frac{\partial L(\{v_k,M_k,\lambda_k\})}{\partial \lambda_k} = \sum_{k=1}^{K} (detM_k - \rho_k)$$

Puisque on a $detM_k = \rho_k$ pour k=1,2..K :

$$\frac{\partial L(\{v_k,M_k,\lambda_k\})}{\partial \lambda_k} = 0$$