MAM 4A

Projet Statistique

Classification en petite et grande dimension

Auteurs :
M. Anwar ABOUABDALLAH
M. Maxime FUCCELLARO

Encadrant :
Mme Lola ETIÉVANT

Table des Matières

1 Introduction

Ce projet de statistiques, mené de février 2018 à juin 2018, est un projet expliquant et illustrant la classification de données. Deux types de classification existent : la classification *supervisée* et la classification *non-supervisée*. Ces deux aspects de la classification constitueront les deux parties majeures de notre rapport.

De nombreux algorithmes -chacun incontournables lorsque l'on veut faire de la classification- ont été des acteurs majeurs de l'essort de la classification dans le domaine des statistiques. Nous détaillerons le fonctionnement de ceux-ci pour comprendre précisément les particularités du *supervisé* et du *non-supervisé* et les différences entres ces deux méthodes de classification.

Une fois les prérequis énoncés et le fonctionnement des algorithmes expliqués, nous illustrerons la classification *supervisée* et *non-supervisée* à travers deux jeux de données distincts. Le premier est le jeu de données dit des "Iris de Fisher" -jeu de données intégré au logiciel *R*- qui sera dit en "petite dimension" du fait de son nombre parcimonieux (4) de variables explicatives concernant la nature de la fleur elle même, c'est à dire *Setosa*, *Versicolor*, ou *Virginica*. Le second jeu de données est accessible sur *R* via le package "spls" et sera dit en grande dimension car il est composé d'énormement de variables explicatives (6034) concernant l'apparition d'un cancer chez un patient.

2 La classification non supervisée

"Le seul moyen de faire une méthode instructive et naturelle, est de mettre ensemble des choses qui se ressemblent et de séparer celles qui diffèrent les unes des autres", écrit Buffon en 1749 dans le Tome 1 de son encyclopédie "Histoire Naturelle". Cette phrase nous révèle la raison première de l'apparition de la classification telle qu'on la connait aujourd'hui : le désir de connaissance. Ce désir a été l'élément déclencheur de la première taxonomie scientifique. Ainsi, la classification devînt le thème central de l'histoire naturelle, puis de la biologie. À la fin du 19ème siècle, la théorie statistique apparaît, ce qui ne fera qu'accroître les performances des méthodes classification, et ce, jusqu'à nos jours.

L'apprentissage non supervisé est une méthode d'apprentissage automatique. Il s'agit de diviser un groupe hétérogène de données en sous-groupes, de manière à ce que les données considérées comme les plus similaires soient associées au sein d'un même groupe (homogène) et qu'au contraire les données considérées comme différentes se retrouvent dans d'autres groupes distincts. Ceci sera effectué sans qu'au préalable la nature des classes ait été choisie ; il s'agit donc d'une classification objective des données.

Définition de l'inertie d'un nuage de points : On dispose de n points $x_1, ..., x_n$ et on désigne par x_G le barycentre du nuage de ces points :

$$x_G = \frac{1}{n} \sum_{i=1}^{n} x_i$$

L'inertie totale est définie comme

$$I_T = \sum_{i=1}^{n} ||x_i - x_G||^2$$

On suppose qu'en réalité le nuage de points est composé de K classes de points distincts $C_1, ...C_k$, chacune de ces classes ayant pour barycentre x_{Ck}. On peut alors décomposer l'inertie totale du nuage de la manière suivante :

$$I_T = \underbrace{\sum_{k=1}^{K} \sum_{i \in C_k} ||x_i - x_{Ck}||^2}_{\textbf{(1)}} + \underbrace{\sum_{k=1}^{K} n_k ||x_{Ck} - x_G||^2}_{\textbf{(2)}} \quad ,$$

où n_k est le nombre d'observations de la classe C_k. Le terme **(1)** mesure la somme des distances entre les points d'une classe et leur barycentre. On appelle cette inertie l'inertie intra-classe. La terme **(2)** mesure à quel point les barycentres des classes sont loin du barycentre global, c'est-à-dire à quel point les classes sont distantes les unes des autres ; c'est l'inertie inter-classe.

Qualité d'une partition obtenue par une classification supervisée :

On dit qu'une partition (découpage des données en groupes) est bonne si deux individus d'une même classe sont proches, ou encore si deux individus de deux classes différentes sont éloignés. Mathématiquement, cela signifie que la variabilité intra-classe est petite et que la variabilité inter-classe est grande. Ces deux critères n'en forment en fait qu'un seul. En effet grâce au théorème de Huygens, on sait que :

$$Inertie\ totale = Inertie\ intra - classe + Inertie\ inter - classe$$

Et comme l'inertie totale est **constante**, minimiser la variabilité intra-classe revient à maximiser la variabilité inter-classe, et vice-versa.
Ceci nous donne un ratio pour évaluer la qualité d'une partition :

$$0 \leq \frac{inertie - inter}{inertie - totale} \leq 1$$

Plus ce ratio sera proche de 1 plus la partition sera bonne.

2.1 Classification Ascendante Hiérarchique (CAH)

Faire une classification, c'est l'action de constituer des ensembles d'individus possédant des traits de caractère communs (appelés "classes"). Deux types de classification sont possibles, la classification hiérarchique d'une part pour laquelle on cherchera à construire un arbre hiérarchique, et les méthodes de partitionnement d'autre part. La classification ascendante hiérarchique est dite ascendante car elle part d'une situation où tous les individus sont seuls dans une classe, puis sont rassemblés en classes de plus en plus grandes. Le qualificatif *hiérarchique* vient du fait que la classification produit une hiérarchie H, l'ensemble des classes à toutes les étapes de l'algorithme, qui vérifie les propriétés suivantes sur un ensemble Ω de n individus :

— $\Omega \in H$
— $\forall \omega \in \Omega, \{\omega\} \in H$
— $\forall (h, h') \in H^2, h \cap h' = \emptyset$ ou $h \subset h'$ ou $h' \subset h$

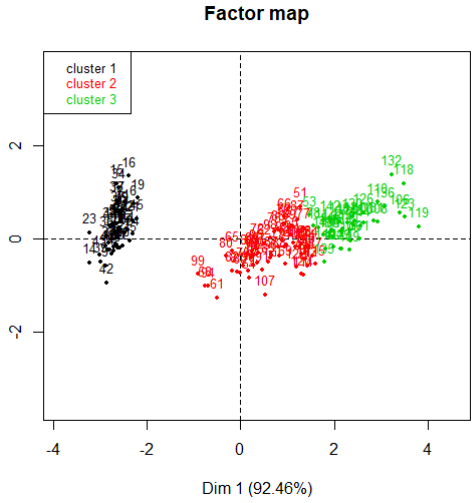
L'objectif cette classification est de produire une arborescence qui met en évidence les liens hiérarchiques entre les individus.

Il est indispensable de définir une mesure de ressemblance entre individus, impliquant que deux individus sont "proches" et peuvent être affectés à une même classe. La plupart du temps, la distance euclidienne est choisie, mais ce n'est pas toujours le choix optimal. Il faut de plus définir la distance entre groupes d'individus. Une première mesure de ressemblance est appelée «saut minimum» ou «lien simple» ; c'est la plus petite distance entre un élément du premier groupe et un élément du second. Il y a aussi le «lien complet» (plus grande distance entre un individu du premier groupe et un individu du deuxième). Le choix de la mesure de ressemblance modifie la classification que l'on obtiendra.

2.1.1 Un exemple en faible dimension

Nous allons d'ores et déjà visualiser les données des iris de Fisher dans le plan des deux premiers axes de l'ACP effectuée sur le jeu de données *iris* :

FIGURE 1 – Visualisation des données d'iris sur les deux premiers axes de l'ACP



Nous constatons que le premier axe de l'ACP concentre plus de 92% de l'information, c'est donc une très bonne représentation des données dans l'espace. Voyons désormais les différents pourcentages des différents types de fleur au sein des trois groupes :

TABLE 1 –

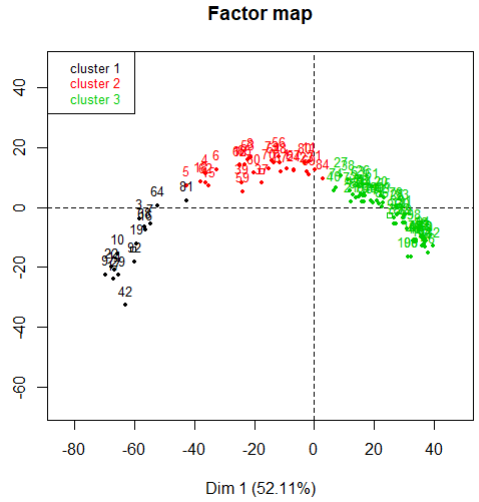
Groupe	% Versicolor	%Setosa	%Virginica
Groupe 1	77%	0%	23%
Groupe 2	5.2%	0%	94.8%
Groupe 3	0%	100%	0%

À partir de ce tableau, on peut dire que l'algorithme de l'algorithme HCPC classifie excellemment bien les iris *Setosa*. En revanche, au vu du pourcentage de *Virginica* dans le groupe 1, on peut constater que ces fleurs sont lus facilement assimilables à des iris *Versicolor*. À moindre mesure, les iris *Versicolor* peuvent être aussi parfois assimilé à des *Virginica* (5.4% dans le groupe 2).

2.1.2 Un exemple en grande dimension

On peut également commencer par regarder la répartition des données de cancer dans le plan formé par les deux premiers axes de l'ACP :

FIGURE 2 – Visualisation des données de cancer sur les deux premiers axes de l'ACP



La représentation de ces données est moyennement fidèle à la réalité au vu du pourcentage d'information concentré sur le premier axe de l'ACP. De plus, on comprend que l'algorithme HCPC a classifier les données en trois groupes et non deux -pour individus saints et individus malades- ce qui peut paraître curieux. Dès lors, voyons le pourcentage de personne ayant le cancer dans ces trois groupes :

TABLE 2 – % cancéreux dans les différents groupes

Groupe	% Cancéreux	Nombre de peronnes
Groupe 1	56%	57
Groupe 2	53%	30
Groupe 3	26%	15

Au vu de ce tableau, on remarque que les deux groupes 1 et 2 se dégagent avec respectivement 57 et 30 personnes au sein de celui-ci.

2.2 Algorithme des k-Moyennes

L'algorithme des K-moyennes a été proposé par Stuart Lloyd en 1957. Le partitionnement en k-moyennes est une méthode de partitionnement de données caractérisée par l'intermédiaire d'un problème d'optimisation combinatoire, et ne nécessite pas la construction d'un arbre hiérarchique. Étant donné des points et un entier k, le problème consiste à diviser les points en k groupes, souvent appelés clusters, de façon à minimiser une certaine fonction. On considère la distance d'un point à la moyenne des points de son groupe ; la fonction à minimiser est la somme des carrés de ces distances.

Définition :

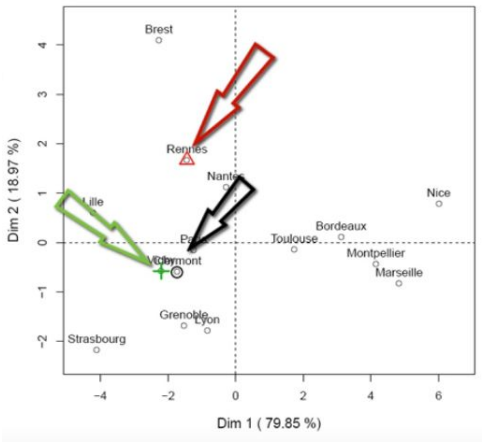
Étant donné un ensemble de points (x_1, x_2,..., x_n), on cherche à partitionner les n points en k ensembles S = ($S_1, S_2, ..., S_k$) en minimisant la distance entre les points à l'intérieur de chaque partition. La fonction à minimiser est la suivante :

$$Arg\underset{S}{Min} \sum_{i=1}^{k} \sum_{X_j \in S_i} ||X_j - \mu_i||^2$$

où μ_i est le barycentre des points dans S_i.

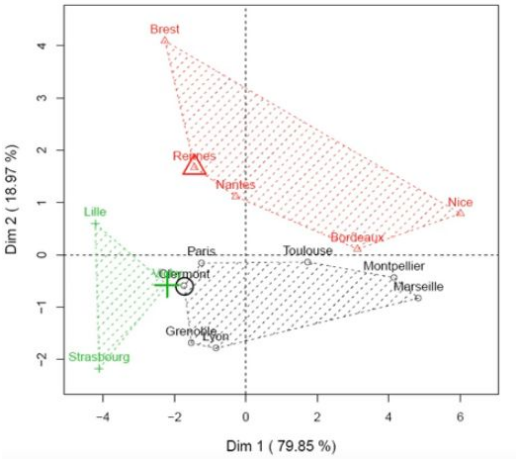
Un exemple, pas à pas : L'algorithme des K-moyennes nécessite de connaître le nombre de classes Q que l'on souhaite construire. Au départ, on choisit Q centres de classes au hasard, c'est-à-dire Q données qui joueront provisoirement le rôle de barycentres des classes ; c'est l'étape d'initialisation de l'algorithme.

FIGURE 3 – Choix au hasard de trois classes



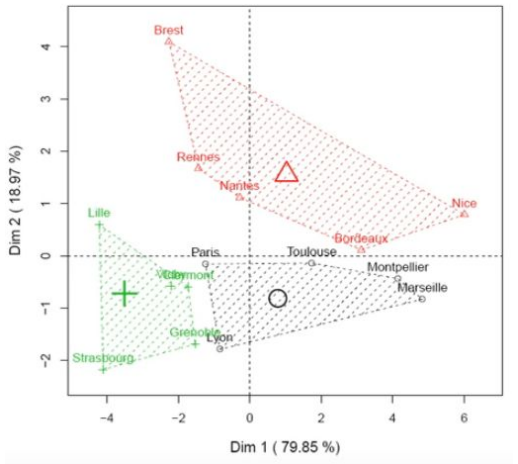
Par exemple, ici, Q=3. Ainsi, Rennes, Vichy et Clermont sont les trois centres de classes tirés au hasard. C'est l'initialisation de l'algorithme. Ensuite, on regroupe tous les points au centre de classe le plus proche.

FIGURE 4 – Premier regroupement des données



Il s'agit désormais de calculer le centre de gravité de chacune des classes, ceux-ci sont cette fois non plus des individus mais bien des points fictifs. Une fois l'opération réalisée, on redistribue les points selon le nouveau centre de gravité :

FIGURE 5 – Redistribution des données



On itère jusqu'à ce que les classes ne changent plus et que les centres de gravité restent fixes. Une fois que cela s'est produit, on dira que l'algorithme a convergé.

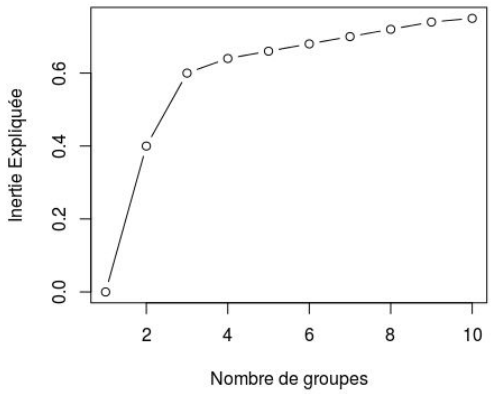
Compléments sur l'algorithme :

L'algorithme des K-means est un algorithme très rapide (complexité polynomiale). Pour autant, cette méthode a deux défauts :

- Le premier est qu'il faut connaître le nombre de classes a priori.
- Le second est que la partition dépend de l'initialisation et du choix des centres de classes au hasard.

Afin de pallier le premier problème on peut se donner un critère, ou encore une heuristique nous permettant de trancher. Deux moyens sont à disposition. Premièrement, la courbe des pourcentages d'inertie expliquée (c'est-à-dire les ratios entre la somme des variances des groupes et la variance totale des données) en fonction du nombre de groupes choisi : l'heuristique ici serait de prendre le nombre de groupes correspondant au "coude" de la courbe, c'est à dire trouver le point à partir duquel les variations de pourcentage d'inertie expliquée ne sont plus très importants. Au vu de la courbe suivante, on pourrait choisir N=3 comme nombre de groupes.

FIGURE 6 – Inertie expliquée en fonction du nombre de groupes

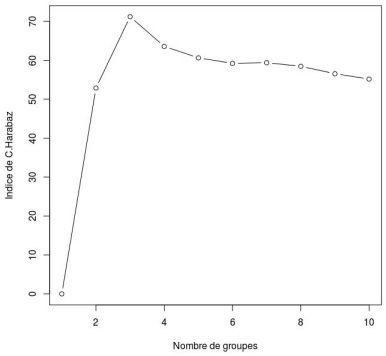


Un second critère nous permettant de trancher est le critère de Calinski Harabaz ; ce critère consiste à maximiser un indice qui varie également selon le nombre de groupes. Il est défini comme suit :

$$\frac{V_{inter}}{V_{intra}} \times \frac{(N-K)}{(K-1)},$$

où V_{inter} est la variabilité inter-classes,V_{intra} la variabilité intra-classe, K le nombre de groupes et N le nombre d'observations. Plus la valeur de ce rapport est élevée, plus l'intérieur de chaque groupe sera homogène (faible variance au sein du groupe) et plus les groupes entre eux seront hétérogènes (variance élevée entre les groupes). L'indice Calinski-Harabasz n'étant pas défini pour $K = 1$, il ne peut pas être utilisé pour détecter le cas où ne choisir qu'un groupe est optimal.

FIGURE 7 – Indice de Calinski Harabaz en fonction du nombre de groupe



Ici, le choix du nombre de groupes est nettement plus évident qu'avec le critère précédent, il sera préférable de choisir trois groupes pour lancer l'algorithme.

Pour remédier au second problème, on lancera plusieurs fois l'algorithme avec des initialisations (c'est-à-dire des centres de classes) différentes.

2.2.1 Un exemple en faible dimension

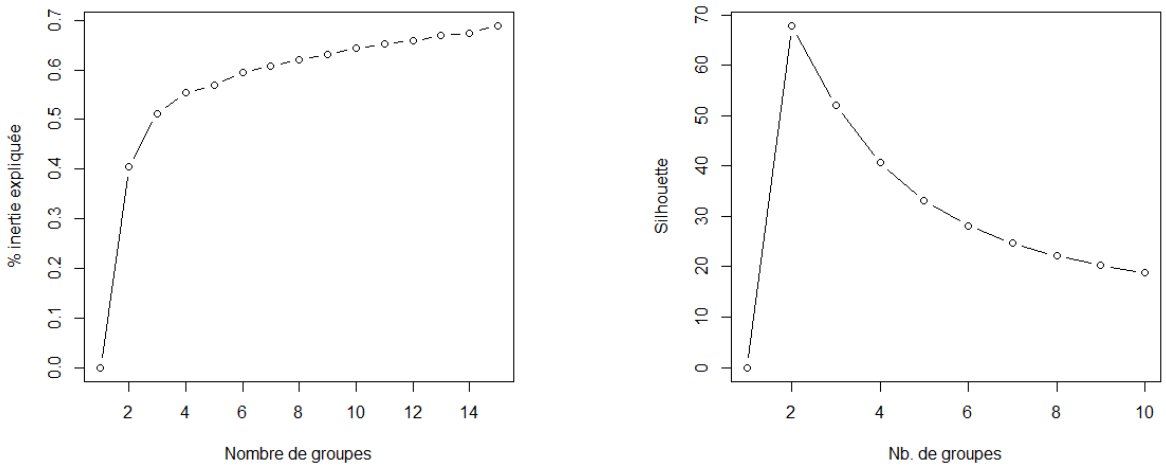
TABLE 3 – Visualisation des % d'iris dans les groupes

Groupe	% Versicolor	%Setosa	%Virginica
Groupe 1	79%	0%	21%
Groupe 2	5%	0%	95%
Groupe 3	0%	100%	0%

Ici, nous obtenons une meilleure classification que pour l'agorithme HCPC.

2.2.2 Un exemple en grande dimension

Regardons tout d'abord l'indice de calinski harabasz et la courbe des inerties expliquées pour trancher sur le nombre de groupe à choisir :



Ici un problème se pose : la courbe des inerties expliquées à plutôt tendance à nous inciter de prendre trois groupes alors que l'indice de calinski harabasz nous invite clairement à en prendre deux. Voyons les pourcentages de personnes cancéreuses dans les deux cas :

TABLE 4 – % Personnes atteintes du cancer dans les différents groupes

Groupe	% Cancéreux
Groupe 1	53%
Groupe 2	26%
Groupe 3	56%

Ici nous obtenons absolument les mêmes résultats qu'avec l'algorithme HCPC, ces deux algorithmes ont donc les mêmes performances pour ce jeu de données.

TABLE 5 – % Personnes atteintes du cancer dans les différents groupes

Groupe	% Cancéreux
Groupe 1	57%
Groupe 2	40%

Nous nous apercevons qu'il y a quasiment le même pourcentage de personnes cancéreuses dans un groupe que dans l'autre. Il semble donc que le fait d'être atteint du cancer ne soit pas un critère distinctif pour cet algorithme.

3 Algorithme E.M. :

3.1 Présentation de l'algorithme :

3.1.1 Présentation générale :

Dans le cas non-supervisé, la maximisation de la log-vraisemblance d'un modèle de mélange peut conduire à des équations de vraisemblance qui ne possèdent pas de solutions analytiques. A dessein de contourner cela, on peut utiliser des algorithmes permettant de maximiser la log-vraisemblance quand les labels sont inconnus. Le plus utilisé d'entre eux est l'algorithme itératif Expectation-Maximization ou espérance maximisation dont le nom est souvent abrégé EM. Cet algorithme a été proposé par Dempster, Laird et Rubin en 1977, il est principalement utilisé en imagerie médicale dans le cadre de la reconstruction tomographique. wiki +these

3.1.2 La construction de l'algorithme :

On suppose que nos observations sont notées $X \in \mathbb{R}^n$ et que nous disposons d'un ensemble de variables latentes Z. L'ensemble des paramètres du modèle sera noté θ. L'objective de notre algorithme E.M. et de déterminer les paramètres qui maximisent la vraisemblance de $p(X|\theta)$ quand ni cette expression ni son logarithme ne peuvent être maximiser par les méthodes de type MLE [1] classiques.
Soit notre jeu de données $\{X, Z\}$ qu'on ne connait pas totalement, on connait seulement les valeurs des $X-_n.$, sa vraisemblance est donnée par $p(X, Z|\theta)$.
Pour cela on construit le postérior $p(Z|X, \theta)$, puis par le biais de $p(Z|X, \theta)$, on peut calculer la vraisemblance du set complet :

$$p(X, Z|\theta) = p(Z|X, \theta)p(X|\theta)$$

On suppose aussi qu'on connait une estimation θ_i pour θ ce qui nous permet de calculer le posterior $p(Z|X, \theta_i)$.
Initialisation :
L'algorithme commence d'abord par faire une assignation aléatoire des valeurs $\theta_i = \theta_0$. Ceci nous permet d'avoir une log-vraisemblance de cette forme :

$$\boxed{\log p(X|\theta) = \log\{\textstyle\sum_z p(X, Z = z|\theta)\} = \log\{\sum_z p(X|Z = z, \theta)p(Z = z|\theta)\}}$$

Pour les variables latentes continues, notre somme sera remplacée par une intégrale, l'algorithme E.M. devrait maximiser $\log p(X|\theta)$ puis comme log est strictement croissante alors l'algorithme E.M. maximisera aussi $p(X|\theta)$. Par la suite, nous allons montrer le fonctionnement des deux étapes de notre algorithme.

À partir de notre estimation θ_i, nous essayerons d'améliorer la valeur de θ afin d'avoir $p(X|\theta) > p(X|\theta_i)$ puis si possible nous maximiseront la différence entre $p(X|\theta)$ et $p(X|\theta_i)$:
Premièrement l'étape E ou E-step :
Cette étape consiste à calculer $E_{Z|X,\theta_i}[\log p(X, Z|\theta)]$ Notre $\theta-_i$ calculé vérifie : $\log p(X|\theta) \geq \log p(X|\theta_i) + \Delta(\theta|X, \theta_i) \equiv Q(\theta|\theta_i)$
Notre nouvelle fonction $Q(\theta|\theta_i)$ est inférieure à $\log p(X|\theta)$ et sont égales en $\theta = \theta_i$.

1. **Méthode MLE :** Méthodes d'estimation par maximum de vraisemblance

$$
\begin{aligned}
& \log p(X|\theta) - \log p(X|\theta_i) \\
= \;& \log\{\textstyle\sum_z p(X|Z=z,\theta)p(Z=z|\theta)\} - \log p(X|\theta_i) \\
= \;& \log\left\{\textstyle\sum_z p(X|Z=z,\theta)p(Z=z|\theta)\frac{p(Z=z|X,\theta_i)}{p(Z=z|X,\theta_i)}\right\} - \log p(X|\theta_i) \\
= \;& \log\left\{\textstyle\sum_z p(Z=z|X,\theta_i)\frac{p(X|Z=z,\theta)p(Z=z|\theta)}{p(Z=z|X,\theta_i)}\right\} - \log p(X|\theta_i) \\
\geq \;& \textstyle\sum_z p(Z=z|X,\theta_i)\log\left\{\frac{p(X|Z=z,\theta)p(Z=z|\theta)}{p(Z=z|X,\theta_i)}\right\} - \log p(X|\theta_i) \\
= \;& \textstyle\sum_z p(Z=z|X,\theta_i)\log\left\{\frac{p(X|Z=z,\theta)p(Z=z|\theta)}{p(Z=z|X,\theta_i)p(X|\theta_i)}\right\} \\
\equiv \;& \Delta(\theta|X,\theta_i)
\end{aligned}
$$

Bilan : E-step : consiste à calculer $Q(\theta|X,\theta_i) = E_{Z|X,\theta_i}[\log p(X,Z|\theta)]$

Deuxièmement l'étape M ou M-step :

Dans cette étape, on souhaite affecter la valeur θ_{i+1} à θ qui maximise $Q(\theta|\theta_i)$.

Comme on a $\log p(X|\theta_{i+1}) \geq Q(\theta_{i+1}|\theta_i)$ alors la vraisemblance va aussi augmenter. Puis montrons $\arg\max_\theta Q(\theta|\theta_i) = \arg\max_\theta \left\{E_{Z|X,\theta_i}[\log p(X,Z|\theta)]\right\}$
On a

$$
\begin{aligned}
\theta_{i+1} &= \arg\max_\theta Q(\theta|\theta_i) \\
&= \arg\max_\theta \left\{\log p(X|\theta_i) + \textstyle\sum_z p(Z=z|X,\theta_i)\log\frac{p(X|Z=z,\theta)p(Z=z|\theta)}{p(Z=z|X,\theta_i)p(X|\theta_i)}\right\} \\
&= \arg\max_\theta \{\textstyle\sum_z p(Z=z|X,\theta_i)\log p(X|Z=z,\theta)p(Z=z|\theta)\} \\
&= \arg\max_\theta \left\{\textstyle\sum_z p(Z=z|X,\theta_i)\log\frac{p(X,Z=z,\theta)p(Z=z,\theta)}{p(Z=z,\theta)p(\theta)}\right\} \\
&= \arg\max_\theta \{\textstyle\sum_z p(Z=z|X,\theta_i)\log p(X,Z=z|\theta)\} \\
&= \arg\max_\theta \left\{E_{Z|X,\theta_i}[\log p(X,Z|\theta)]\right\}
\end{aligned}
$$

Bilan : M-step : consiste à calculer $\theta_{i+1} \leftarrow \text{argmax}_\theta\, Q(\theta|X,\theta_i)$

Enfin il reste à déterminer une condition d'arrêt pour notre algorithme, pour cela on va choisir arbitrairement un ϵ petit, puis on continue l'algorithme jusqu'à ce que cette condition soit réalisée :

$$\|\theta_i - \theta_{i+1}\| > \epsilon$$

3.2 Pseudo-code :

Ensuite, à partir de cela on va établir un pseudo-code pour l'algorithme E.M., on a choisit de le faire via une boucle "if" bien qu'il est possible de le faire en boucle "while", de plus on a choisit d'utiliser un epsilone comme condition d'arrêt. Durant notre projet on a utilisé la

Algorithm 1 Algorithme EM

Require: X, $p(Z|X,\theta)$, ϵ
  $\theta_i \leftarrow \theta_0$, ou θ_0 est une affectation aléatoire de valeurs.
  E-step :
  $Q(\theta|X,\theta_i) = E_{Z|X,\theta_i}[\log p(X,Z|\theta)]$
  M-step :
  $\theta_{i+1} \leftarrow \text{argmax}_\theta\, Q(\theta|X,\theta_i)$
  if $\|\theta_i - \theta_{i+1}\| > \epsilon$ **then**
    $\theta_i \leftarrow \theta_{i+1}$
  end if

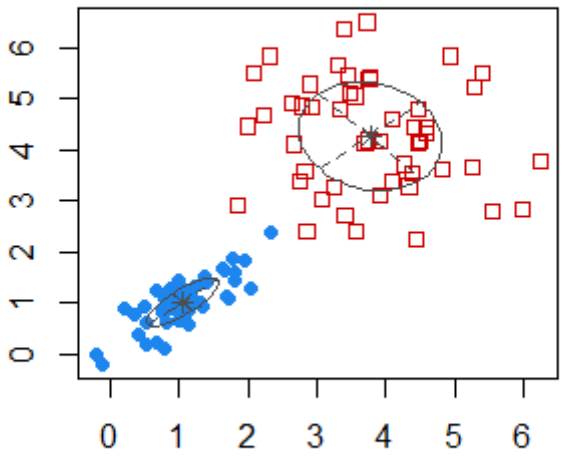
librairie Mclust de R crée en 2017, un choix dut à sa facilité de manipulation.

3.3 Application sur un petit jeu de données :

3.4 Application sur les données du projet :

3.4.1 Les iris de Fisher : Faible dimension

Le jeu de données Iris connu aussi sous le nom de Iris de Fisher est un jeu de données multivariées, il comprend 50 échantillons de chacune des trois espèces d'iris. Nous avons choisi d'appliquer l'algorithme E.M. sur ce jeu de donné, voici le résultat.



A partir de ce graphique on peut remarquer deux classes ou clusters, la première en bleu et la seconde en rouge. En plein milieu de chaque classes se trouve le résultat de l'e.m. sur les densité qui peut être vu en choisissant l'option "density" qui permet de modéliser la fonction qui donnes les probabilités d'appartenir à un cluster.

Ensuite on effectue un summary afin d'avoir toutes les informations sur les cluster ce qui nous permet d'avoir :

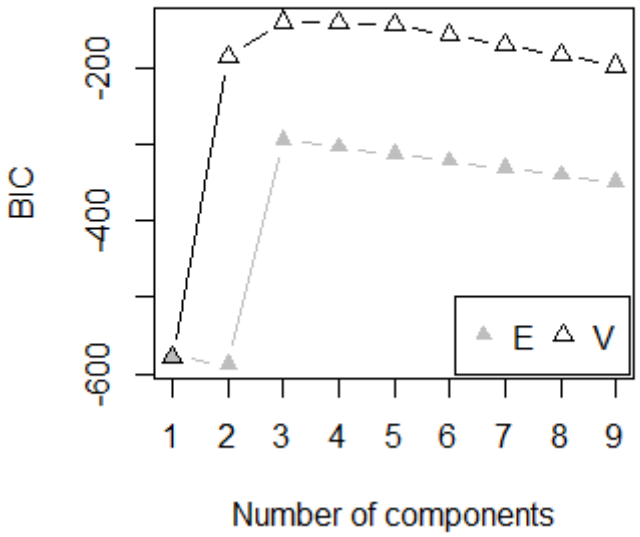
Groupe	% de fleurs
Groupe 1	50%
Groupe 2	50%

Ce résultat montre la seule faiblesse de l'algorithme e.m., il n'est optimisé pour les données latentes, donc il ne distingue pas les noms des fleurs et donc n'apporte aucune information sur leur espèces : setosas, versicolor ou virginica.
En effet le second groupe rassemble à la fois l'Iris virginica et l'Iris versicolor.

3.4.2 Les données prostate : Grande Dimension

Le second jeu de donnée utilisé est issu des résultats du livre "The Elements of Statistical Learning - 2nd Edition" en testant les modèles sur les données de cancer de la prostate. Il est disponible dans les packages utilisés sur R.

Ici, nous avons utilisé une Acp afin de réduire la dimension de nos données, ensuite nous avons appliqué l'algorithme em sur nos données ce qui a permis d'avoir une courbe d'inertie de la forme :



Cette courbe nous indique l'existence de trois clusters, cette indication est confirmée par le sommaire de "Mclust".

Groupe	% Cancéreux
Groupe 1	58%
Groupe 2	38%
Groupe 3	4%

Variantes d'EM

Malgré l'efficacité de l'algorithme EM, certaines problématiques ont donné naissance à trois variantes connues, l'algorithme GEM (generalized EM) qui permet de simplifier le problème de l'étape de maximisation ; l'algorithme CEM (classification EM) permettant de prendre en compte l'aspect classification lors de l'estimation, ainsi que l'algorithme SEM (stochastic EM) dont l'objectif est de réduire le risque de tomber dans un optimum local de vraisemblance.

4 Classification supervisée :

Dans un premier temps, on va se donner des classes "a priori" pour effectuer la classification supervisée. Par exemple, deux classes possibles pour un jeu de données seraient "cette donnée correspond à un patient saint" et "cette donnée correspond à un patient atteint d'une tumeur". Il s'agit donc d'une classification subjective des données, celle-ci passant par une phase d'apprentissage et une phase de validation. Pour Michalski, "Apprendre consiste à construire et à modifier des représentations ou des modèles à partir d'une série d'expériences". Ainsi, lors de la phase d'apprentissage, on va chercher à construire des règles de décision concernant les différentes classes prédéfinies via une sous partie du jeu de données initial, appelé données d'apprentissage.

La méthode d'apprentissage supervisé utilise cette base d'apprentissage pour déterminer la classe x' d'une certaine donnée d'entrées x. Le but d'un algorithme d'apprentissage supervisé est donc de généraliser ce qu'il a pu « apprendre » grâce aux données déjà traitées (données d'apprentissage). Lorsque la sortie que l'on cherche à estimer est une valeur dans un ensemble continu de réels, on parle d'un problème de régression. La fonction de prédiction est alors appelée un régresseur. Lorsque l'ensemble des valeurs de sortie est fini, on parle d'un problème de classification, qui revient à attribuer une étiquette à chaque entrée. La fonction de prédiction est alors appelée un classificateur.

Que l'on se trouve dans le cas supervisé ou non-supervisé il faut bien choisir son modèle afin de ne pas sur ou sous apprendre. Un exemple de sur apprentissage est une interpolation polynomiale passant par un grand nombre de point. Le polynôme ainsi obtenu ne s'adaptera pas a de nouvelles données. Au contraire, si le polynôme ne passait par aucun point, alors celui-ci ne s'adaptera pas à de nouvelle données. D'où l'importance d'avoir un bon modèle.

4.1 Comment choisir son modèle ?

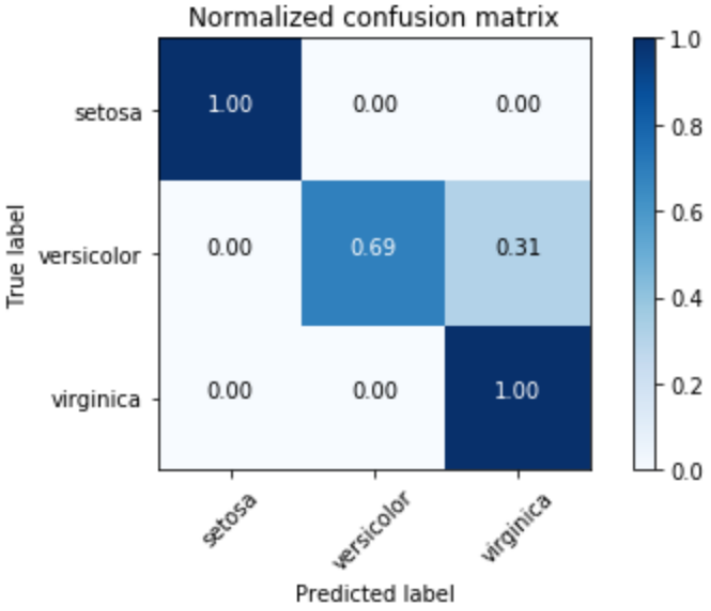
Afin d'avoir un modèle qui se généralise bien on va procéder a une validation croisée. Le principe est simple. On va procéder a une séparation de la base de données en deux partie, "train" et "test". On va ensuite lancer un apprentissage sur la partie train et évaluer la qualité de celui-ci grâce à plusieurs métriques d'évaluation tel que l'AUC ou la matrice de confusion que l'on détaillera par la suite. On va répéter le processus un grand nombre de fois et, si les métriques indiquent un bon modèle et sont stables suivant les itérations, on a affaire à un bon modèle. Sinon, le modèle est à revoir, on peut souvent modifier les paramètres de l'algorithme afin d'améliorer celui-ci.

4.2 Métriques d'évaluations

4.2.1 Matrice de confusion

La matrice de confusion est une matrice carrée qui a autant de lignes que le problème a de modalités dans la cible. Dans le cas d'une classification binaire, cette matrice est donc de taille 2x2. Dans le cas des iris, la matrice sera donc de taille 3x3.

FIGURE 8 – Matrice de confusion obtenue à partir d'une régression logistique



En abscisse on a ce que l'algorithme a prédit alors qu'en ordonnée on a les vrais label. Sur la diagonale on a donc le pourcentage de label qui ont été correctement prédit par espèce.

4.2.2 AUC

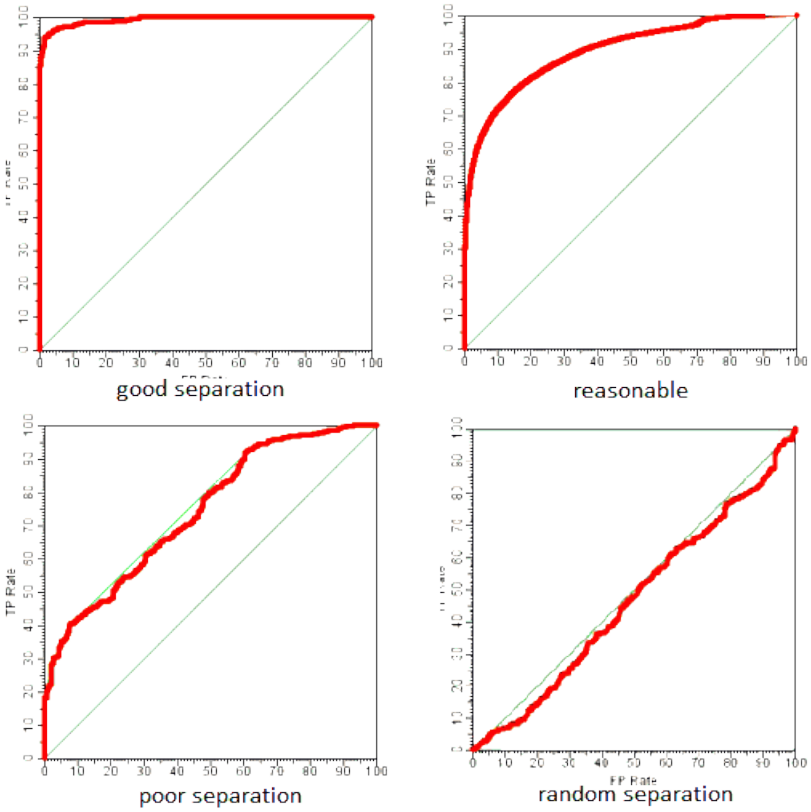
Afin d'évaluer notre modèle nous allons utiliser l'Area Under Curve (AUC), comme métrique d'évaluation du modèle, c'est l'aire en dessous de la courbe ROC (Receiver operating characteristic). Elle est automatiquement calculée par de nombreuses librairies *python*. L'AUC est une métrique de classification binaire, mais elle peut être adaptée à la classification multi-classes. Cette mesure permet d'estimer la qualité les prédictions d'un algorithme par rapport à une prédiction aléatoire.

Pseudo-code pour calculer l'AUC : considérons que l'on souhaite calculer l'AUC pour un score de probabilité d'une base de test de taille n.

1. Récupérer le vecteur $\vec{p}$ de probabilité d'obtention de la cible 1 du jeu de test produite par le modèle, ainsi que les vraies valeurs $\vec{y}$ prise par la cible.
2. Ordonner les probabilités par ordre décroissant.On a : $p_0 >= p_i >= p_{i+1} >= p_n$.
3. Réordonner la cible pour avoir p_i la probabilité que $y_i = 1$
4. On va maintenant tracer la courbe ROC, pour Receiver Operating Characteristic. Pour chaque probabilité p_i :
   (a) Si $y_i = 1$ tracer un segment vertical d'unité 1.
   (b) Sinon tracer un trait horizontal d'unité 1.
5. L'AUC est l'aire sous la courbe ainsi tracé.

Ainsi l'AUC d'un score parfaitement classifié est de 1, et d'un score aléatoire est autour de 0.5. Si l'AUC prend une valeur inférieure à 0.5 alors les prédictions sont moins bonne que l'aléatoire.

FIGURE 9 – Exemple de courbe ROC



4.3 Forêts aléatoires

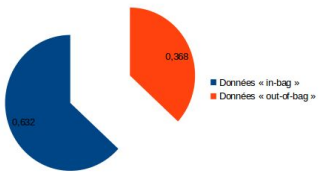
Explication de ce que sont les random forest

Les forêts aléatoires, qui sont un exemple de méthode de classification supervisée, ont été formellement proposées en 2001 par Leo Breiman et Adèle Cutler. Cette méthode engendre une multitude d'arbres de décision où chacun d'eux contribuera finalement à la création de la forêt aléatoire. On va s'intéresser à la création d'une forêt aléatoire dans le cas où la classe x' asssociée à la donnée d'entrée x est discrète (arbres de classification).

Différentes étapes de l'algorithme permettant de créer un arbre de décision :

— Tout d'abord, pour chaque arbre, il faudra scinder le jeu de données de taille n en deux parties de manière aléatoire. La première partie, appelée données "in-bag" [2] correspond à 63.2% des données et l'autre, appelée données "out-of-bag" [3] correspond à 36.8% des données.

FIGURE 10 – Tri aléatoire des données



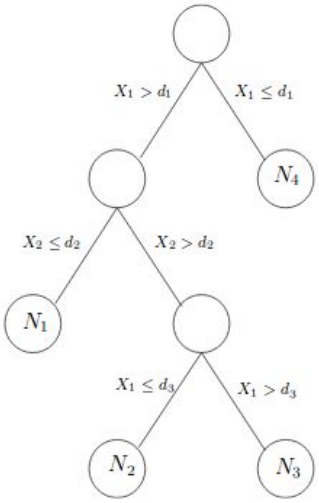
— Ceci effectué, on laisse les données "out-of-bag" de côté et on récupère une fraction (une nouvelle fois 63.2%) des données "in-bag".

— De cette fraction des données découlera le premier arbre de décision : On part d'une variable séparatrice intiale X_1 propre au jeu de données et d'un réelle d pour scinder en deux parties [4] la fraction des données "in-bag".

2. Les données "in-bag" seront les données d'apprentissage
3. Les données "out-of-bag" diffèrent d'un arbre décisionnel à un autre
4. Par exemple, un critère pourrait être " Température $>$ 25°C ?" Si oui, on prend le chemin de gauche, si

— On va répéter ce procédé en prenant à chaque noeud de l'arbre la variable séparatrice X_i et le réel d qui maximisent l'indice d'impureté de Gini. [5]

FIGURE 11 – Présentation d'un arbre de classification de la forêt aléatoire (4 données ici)



— On itère ce procédé jusqu'à ce qu'il n'y ait plus qu'une seule donnée sur chaque feuille de l'arbre de décision, ou encore lorsque toutes les données d'un noeud sont de la même classe (ce sont les deux critères d'arrêt).
— Dès lors nous obtenons une multitude de chemins menant chacun à une ou plusieurs données, chacune renfermant la même classe de sortie (donnée relative à un impact de foudre ou non). La prochaine étape consiste à lancer les données "out-of-bag" dans l'arbre qui vient d'être créer par la fraction des données "in-bag", et vérifier via un vote à la majorité si sur chaque feuille de l'arbre les données "out-of-bag" conduisent à la même conclusion que la fraction des données "in-bag" concernant leur classe de sortie.

Erreur Out Of Bag : Comme on l'a dit plus haut, chaque arbre est entraîné sur une fraction des données, qui est considérée comme « in-bag ». Ce qui permet à chaque arbre, une fois construit, d'estimer son taux d'erreur sur les données qu'il a laissées « out-of-bag » : plus ce taux est faible, plus le modèle est bien ajusté. Ce chiffre à lui seul pourrait servir d'indicateur de performance du modèle.

Voici les points essentiels à retenir des forêts aléatoires :

— 500 sous-ensembles de nos données sont tirés aléatoirement, avec remise. Ces échantillons représentent chacun 63.2% de l'ensemble (500 est un nombre par défaut que l'on peut régler).
— Pour chaque sous-ensemble, un arbre de décision est créé.
— N = Nombre de variables explicatives. Pour chaque arbre, on va retenir $\sqrt[2]{N}$ variables de segmentation de manière aléatoire, et l'arbre sera séparé en suivant la meilleure segmentation (selon l'indice de Gini).
— Lorsque des données "out of bag" sont présentées au modèle, elles seront évaluées par tous les arbres.

Paramètres supplémentaires pour jouer sur la précision et le temps de calcul (fonction RandomForest() sur R) :

— *ntree* : Nombre d'arbre créés

non, celui de droite.

5. Le concept de pureté fait référence au caractère discriminant de la séparation effectuée par un noeud. Une séparation est dite "pure" quand chaque partie post-séparation contient des éléments d'une même classe.À l'inverse, le maximum d'impureté est atteint lorsque chaque séparation contient la même probabilité d'éléments de chaque classe. Soit un ensemble de classes $\{1, ..., L\}$, l'indice de Gini d'un noeud t est défini par $\phi(t) = \sum_{c=1}^{L} p_t^c(1 - p_t^c)$ où p_c^t est la proportion d'observations de classe c dans le noeud t.

— *mtry* : le nombre de variables testées à chaque segmentation. La valeur par défaut étant la racine carrée du nombre de variables explicatives. C'est le principal paramètre à modifier, avec ntree.
— *subset* : Pour ne travailler que sur certaines parties du jeu de données.
— *replace* : Le rééchantillonnage est-il fait avec ou sans remise ?
— *importance* : le modèle doit-il évaluer l'importance des variables ? (Valeur de l'indice de Gini pour chaque variable).

4.4 Algorithme de classification : Régression logistique

La régression logistique est un modèle de régression non-linéaire utile pour mesurer l'association entre la venue d'un évènement Y et les facteurs X_i $(i = 1, ..., p)$, susceptibles de l'influencer. La variable Y est une variable qualitative **binaire**, comme par exemple l'apparition d'une maladie (oui/non) ou encore pour l'apparition d'un impact de foudre (oui/non). Le caractère dichotomique (binaire) de cette variable fait en sorte que la distribution de probabilité envisagée est celle de Bernoulli. Dans ce contexte, l'une des modalités de Y constitue alors le "succès" (codé 1) et l'autre, l'"échec" (codé 0). Ce type de modèle permet de calculer des prévisions sur Y à partir des valeurs des variables X_i (supposées déterministes).

On considère le cas général avec n observations et p variables indépendantes $X_1, ..., X_p$. Pour chaque observation $(Y_i, X_{i1}, ..., X_{ip})$, $i = 1, ..., n$, Y_i suit une loi de Bernoulli de paramètre θ_i (=la probabilité du "succès"). Compte tenu de la distribution de probabilité, on a :

$$E(Y_i) = \theta_i \quad \text{et} \quad Var(Y_i) = \theta_i(1 - \theta_i)$$

Pour autant, le modèle de régression de la forme

$$Y_i = \beta_0 + \beta_1 X_{i1} + ... + \beta_p X_{ip} + \epsilon_i, i = 1, ..., n \quad ,$$

pose un problème ici car $0 \leq E(Y_i) \leq 1$, $Var(Y_i)$ dépend de $E(Y_i)$ et n'est pas constante. Précisément,

$$Y_i = E(Y_i) + \epsilon_i, \quad avec \quad E(\epsilon_i) = 0, \quad 1, ..., n \quad (1)$$

où $E(Y_i) = g(X_{i1}, ..., X_{ip}, \beta)$ est la fonction de régression. Il faut donc trouver une fonction de régression convenable. Si on considère une fonction de régression linéaire :

$$g(X_{i1}, ..., X_{ip}, \beta) = \beta_0 + \sum_{j=1}^{p} X_{ij}\beta_j = \mathbf{X}_i\beta \ , \qquad (2)$$

la représentation est inadéquate car il faudrait que $0 \leq g(X_{i1}, ..., X_{ip}, \beta) \leq 1$ quelles que soient les $X_{i1}, ..., X_{ip}$.

La solution envisagée consiste à lier la combinaison linéaire (2) à $E(Y_i)$ par une fonction bijective dont l'image est dans l'intervalle [0,1]. Un des choix utiles est la fonction "logit" définie par :

$$\eta = \ln(\frac{\theta}{1-\theta}) \ , \quad 0 < \theta < 1$$

$$et \quad \theta = \frac{\exp(\eta)}{1 + \exp(\eta)} = \frac{1}{1 + \exp(-\eta)}$$

FIGURE 12 – Le graphe de θ en fonction de η

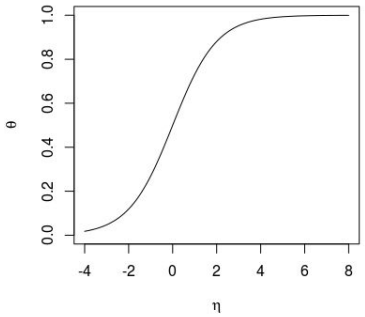
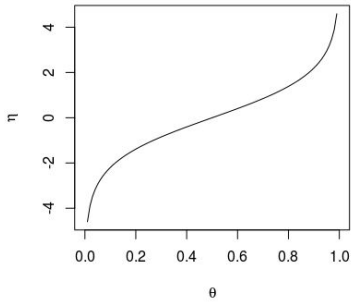


FIGURE 13 – Le graphe de η en fonction de θ



En posant $\eta = \mathbf{X}_i\beta$, il vient que

$$\ln(\frac{\theta_i}{1-\theta_i}) = \beta_0 + \sum_{j=1}^{p} X_{ij}\beta_j = \mathbf{X}_i\beta$$

c'est-à-dire

$$\theta_i = E(Y_i) = \frac{\exp(\mathbf{X}_i\beta)}{1+\exp(\mathbf{X}_i\beta)} = \frac{1}{1+\exp(-\mathbf{X}_i\beta)}, i = 1, ..., n$$

ce qui donne à la fonction de régression une forme cohérente avec l'équation (1).

Dans le cas d'une seule variable explicative (p=1), supposons que l'on dispose de n observations de la forme $(X_1, Y_1), ..., (X_n, Y_n)$, où les Y_i forment une suite de 1 et de 0 selon que le "succès" est observé ou pas. Le modèle est alors de la forme

$$\theta_i = \frac{exp(\beta_0 + \beta_1 X_i)}{1 + exp(\beta_0 + \beta_1 X_i)}, \quad i = 1, ..., n \quad (3)$$

Il en découle que :

$$\frac{\theta_i}{1-\theta_i} = exp(\beta_0 + \beta_1 X_i) \quad et \quad \ln(\frac{\theta_i}{1-\theta_i}) = \beta_0 + \beta_1 X_i, \quad i = 1, ..., n$$

Les paramètres β_0 et β_1 sont estimés par la méthode du maximum de vraisemblance.

La vraisemblance : On considère que pour $i = 1, ..., n$, Y_i suit une loi de Bernoulli de paramètre θ_i donné par l'équation (3). La fonction de masse de Y_i est alors

$$p_i(Y_i) = \theta_i^{Y_i}(1-\theta_i)^{1-Y_i}$$

Pour les n observations la fonction de masse conjointe de $Y_1, ..., Y_n$ est

$$p(Y_1, ..., Y_n) = \prod_{i=1}^{n} p_i(Y_i) = \prod_{i=1}^{n} \theta_i^{Y_i}(1-\theta_i)^{1-Y_i}$$

Le logarithme de la vraisemblance $L(\beta_0, \beta_1)$= $\ln(p(Y_1, ..., Y_n))$ est

$$L(\beta_0, \beta_1) = \sum_{i=1}^{n} Y_i \ln(\theta_i) + \sum_{i=1}^{n} (1-Y_i)\ln(1-\theta_i)$$

$$L(\beta_0, \beta_1) = \sum_{i=1}^{n} Y_i \ln(\frac{\theta_i}{1-\theta_i}) + \sum_{i=1}^{n} \ln(1-\theta_i)$$

$$L(\beta_0, \beta_1) = \sum_{i=1}^{n} Y_i(\beta_0 + \beta_1 X_i) - \sum_{i=1}^{n} \ln(1 + exp(\beta_0 + \beta_1 X_i))$$

Par définition, les éstimateurs de β_0 et β_1 par la méthode du maximum de vraisemblance sont les valeurs qui maximisent $L(\beta_0, \beta_1)$. On les obtient en résolvant le système d'équation suivant :

$$\begin{cases} \frac{\partial L(\beta_0,\beta_1)}{\partial \beta_0} &= \sum_{i=1}^{n} Y_i - \sum_{i=1}^{n} \frac{exp(\beta_0+\beta_1 X_i)}{1+exp(\beta_0+\beta_1 X_i)} = 0 \\ \frac{\partial L(\beta_0,\beta_1)}{\partial \beta_1} &= \sum_{i=1}^{n} X_i Y_i - \sum_{i=1}^{n} \frac{X_i exp(\beta_0+\beta_1 X_i)}{1+exp(\beta_0+\beta_1 X_i)} = 0 \end{cases}$$

Ce système d'équations non linéaires ne donne pas lieu à une solution explicite. Pour autant, des logiciels de statistique (comme R) fournissent des estimations de ces coefficients via des algorithmes d'optimisation numériques (ex : Newton-Raphson).

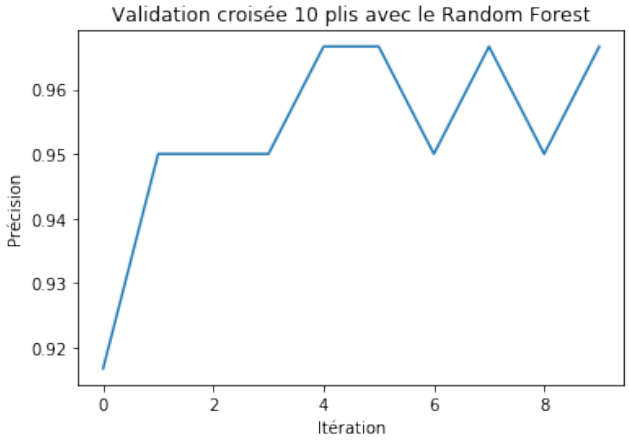
4.4.1 Un exemple en faible dimension : Les iris de Fischer

Pour la régression logistique, il faut d'abord adapter l'algorithme a ce problème à 3 modalités différentes. Car normalement, l'algorithme ne permet que de faire des prédictions du type : L'observation X est de type X, ou l'observation n'est pas de type X.
Pour remédier à cela on va appliquer la technique *leave one out*, voici la procédure :
Pour notre dataset des iris, nous avons 3 modalités différentes. On va donc d'abord prédire si les fleurs sont d'espèces : setosa, puis si elle sont d'espèce versicolor, puis si elle sont d'espèce virginica.

On va donc transformer la variables cible trois fois adéquatement, et prédire 3 fois de suite si une observation appartient aux trois espèces différentes. On va retenir le score ayant la probabilité la plus haute. Cette adaptation est disponible en annexe. Naturellement une implémentation canonique de cette méthode existe déjà.

Résultats de l'algorithme des Forêts aléatoires sur le jeu de donnée des iris.

FIGURE 14 – Résultats de l'algorithme des Forêts aléatoires



On voit ici que le score (Vrai positif) est stable à travers les itérations de la validation croisée. Le score est notamment excellent, on classifie en moyenne avec une précision supérieure à 95%. Les résultats dépassent en moyenne ceux obtenus par régression logistique.

4.5 Un exemple en grande dimension

Présentation du Dataset : Contrairement au dataset des iris, ce jeux de données comporte un très grand nombre de colonnes pour un faible nombre d'individus. En effet, il est composé de 102 observations de 6 034 colonnes et d'une variable cible. L'information peut donc être comme "dilué".

On peut effectuer certains traitements préalable afin de voir si l'on peut concentrer l'information pour augmenter les performances des algorithmes. En d'autres mots, on cherche a voir

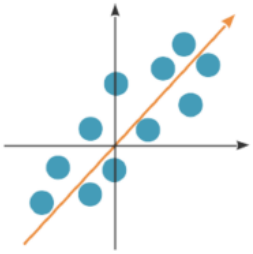
si on peut réduire le nombre de variable sans perdre trop d'information. Ici, on effectuera une Analyse en composante principale où ACP.

4.6 Principe de fonctionnement d'une ACP

L'objectif : maximiser la variance tout en réduisant la dimension.

Le but d'une analyse en composantes principales est de trouver une nouvelle base orthonormée dans laquelle représenter nos données, telle que la variance des données selon ces nouveaux axes soit maximisée.

FIGURE 16 – Variance des données



La variance des données selon l'axe orange est grande. Si on projette les points sur cet axe, ils auront tous des coordonnées différentes ; en utilisant cet axe comme unique dimension, on

réduit la dimension de nos données (de 2 à 1) mais on continue à pouvoir distinguer les points les uns des autres. Calculer les composantes principales

Supposons que nous ayons p variables : chaque observation est représentée par un vecteur dans $\mathbb{R}_p$, et nous avons n observations rassemblées dans une matrice $X \in \mathbb{R}_{p\times n}$.
Commençons par chercher une nouvelle direction, à savoir un vecteur $w1\mathbb{R}_p$. de norme 1, tel que la variance de nos données projetées sur cette direction soit maximale. La projection des données X sur w est noté z1.

Dans ce qui suit, nous allons supposer que X est centrée, c'est-à-dire que la moyenne de ses colonnes est 0.
La variance de $w1 \times X^T$ est égale à $\mathbb{E}[(w1^T X[w^T X])2]$, soit, comme $\mathbb{E}[X] = 0$ (les données sont centrées), $w1^T \mathbb{E}[XX^T]w1$.

Appelons Σ la matrice de taille $p\times p$ égale à XX^T. C'est la matrice de covariance des données et une estimation de $\mathbb{E}[XX^T]$ Nous cherchons donc w1 tel que $w1^T \times \Sigma \times w1$ est maximale et que $||w1|| = 1$
Σ est, par construction, une matrice symétrique définie positive. Elle est donc diagonalisable par un changement de base orthonormé : $\Sigma = Q^T \Delta Q$, où $\Delta \in \mathbb{R}_{p\times p}$ est une matrice diagonale dont les valeurs diagonales sont les valeurs propres de Σ, qui sont toutes positives.

Nous avons donc $w_1^T \Sigma w_1 = w_1^T Q^T \Delta Q w_1 = (Qw_1)^T \Delta (Qw_1)$

Posons $v = Qw_1$; nous cherchons à maximiser $\Sigma_{j=1}^p v_j^2 \lambda_j^2$. Comme ||w1||=1 et que Q est orthonormée, ||v||=1 et donc $\Sigma pj = 1v2j = 1$. La somme $\Sigma_{j=1}^p v_j^2 \lambda_j^2$ est donc maximisée pour un vecteur v qui a une seule entrée à 1 et les autres à 0, l'entrée à 1 étant pour la valeur maximale des λ_j, autrement dit la plus grande valeur propre de Σ.

Par conséquent, w_1 est le vecteur propre correspondant à la plus grande valeur propre de Σ. C'est la première composante principale de X.

La deuxième composante principale de X doit vérifier les mêmes critères que w_1 : être de norme 1 et maximiser $w_2^T \Sigma w_2$, mais lui est orthogonale. Il s'agit donc du vecteur propre de Σ correspondant à sa deuxième plus grande valeur propre.

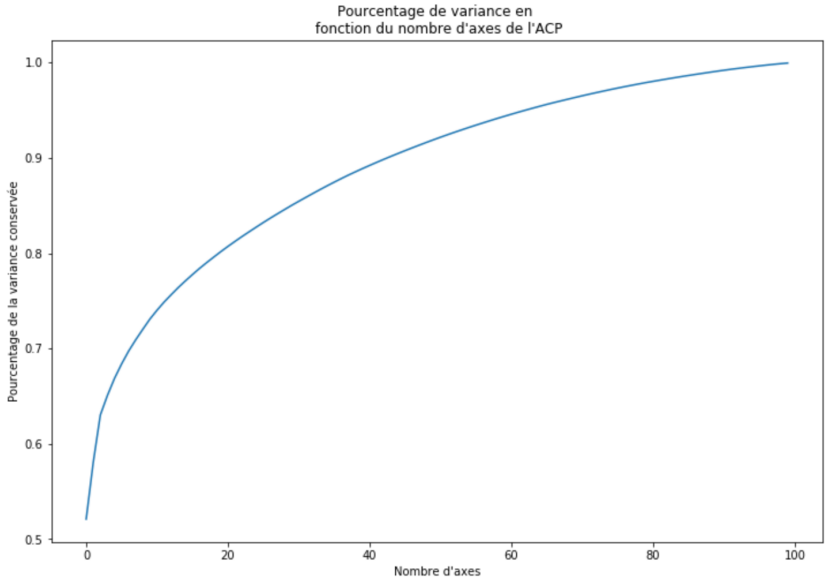
Et ainsi de suite pour les autres composantes principales.

Nous pouvons donc utiliser comme nouvelles dimensions la base formée par les vecteurs propres de la matrice de covariance des données.

Nous allons évaluer les performances des algorithmes présentés précédemment dans un contexte totalement différent, celui de la grande dimension.

Pour sélectionner le nombre d'axe optimal dans notre jeu de donnée du cancer et juger de la pertinence de l'ACP on va regarder le graphique suivant :

FIGURE 15 – Variance des données



On remarque que les 100 premiers axes de notre acp intègre 99% de la variance. Les noms des variables n'ayant pas de signification métier, on va donc remplacer nos variables par les axes de l'ACP.

Regardons maintenant les résultats de nos deux algorithmes sur les données transformées.

4.6.1 Résultat de l'algorithme des Forêts Aléatoires

FIGURE 16 – Courbe Roc des prévisions

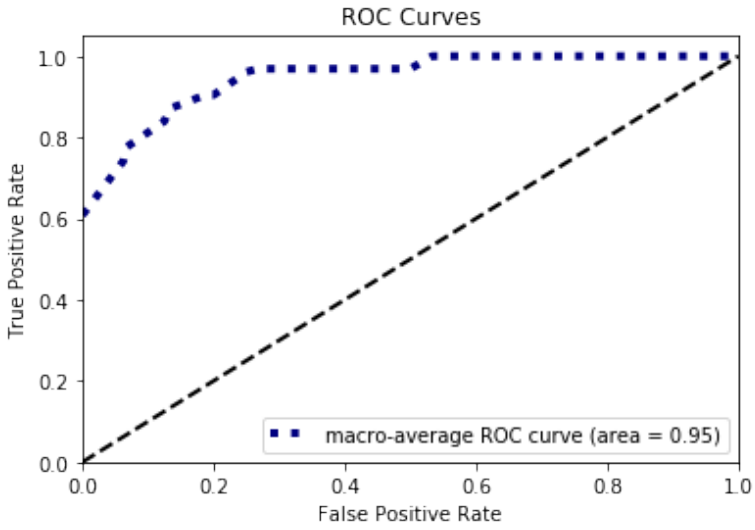
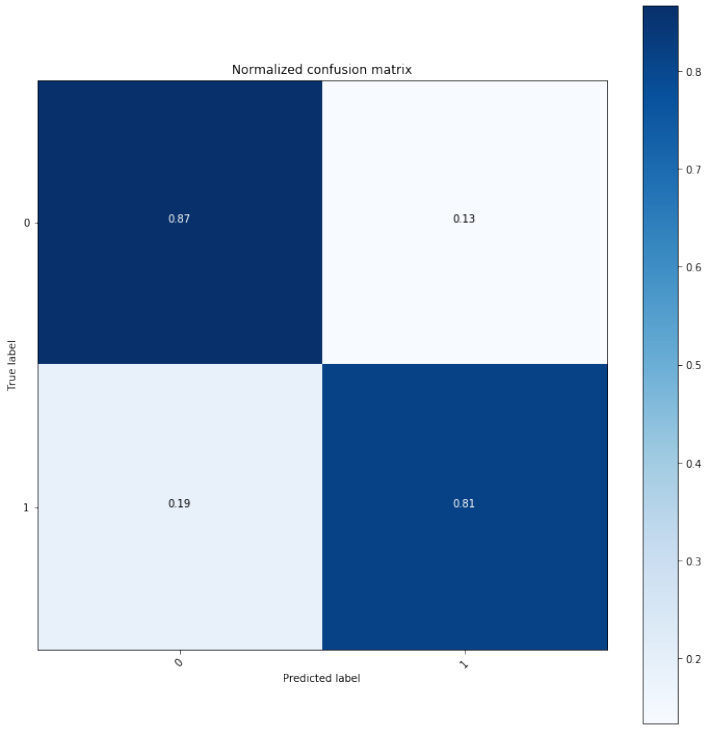


FIGURE 17 – Matrice de confusion des prévisions



Ces deux types de scores nous montrent en effet que l'algorithme est d'une grande efficacité. La courbe Roc et l'AUC nous indiquent que les résultats sont très biens classés. La matrice de confusion nous donne un taux de vrai positifs et vrai négatifs supérieur à 80%.

4.6.2 Résultat de l'algorithme Régression Logistique

FIGURE 18 – Courbe Roc des prévisions

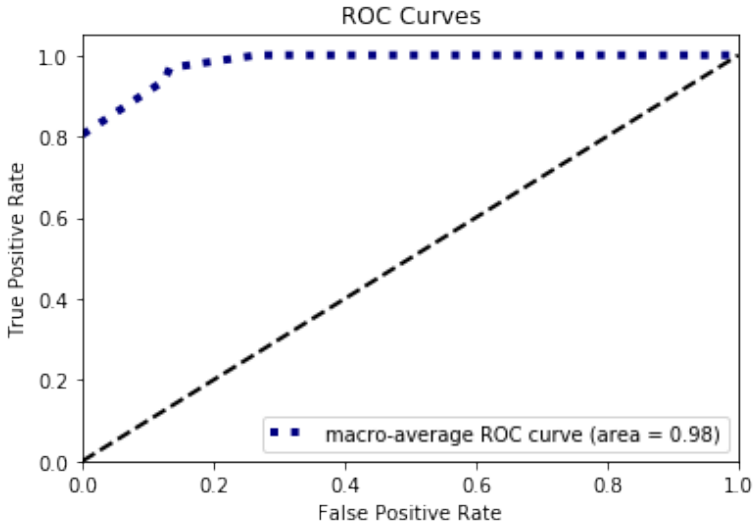
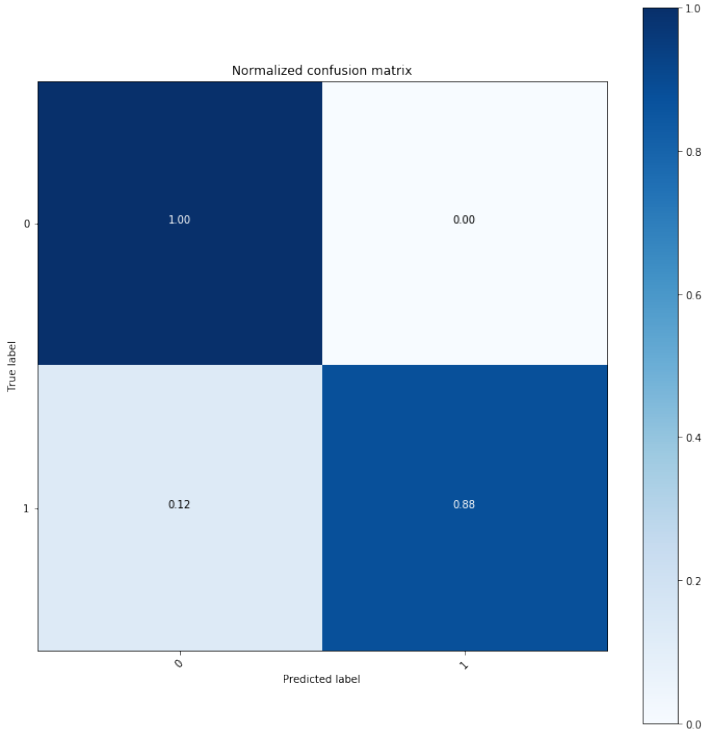


FIGURE 19 – Matrice de confusion des prévisions



Ces deux types de scores nous montrent en effet que la regression logistique est encore plus éfficace que le Random Forest. L'AUC proche de 1 nous indique une presque parfaite séparation des données. La matrice de confusion nous confirme les résultats de l'AUC en indiquant une parfaite évaluation des vrais positif. Le taux de faux positif de 12% est remarquable et pourrait très bien être amélioré par d'autres techniques tel que le re-sampling qui consiste a ajuster le taux de positif dans la base d'entrainement afin de maximiser les performances des algorithmes.

5 Conclusion

À travers ce projet, nous avons pu apprendre plusieurs manières d'appréhender les données selon ce que l'on compte faire sur celles-ci. En effet, si notre but est de rassembler les données en groupes homogènes, alors nous utiliserons notre savoir sur la classification non supervisée. En revanche, si nous voulons faire des prévisions sur une variable à expliquer à partie d'un jeu de données composé de variables explicatives, il faudra utiliser nos connaissances sur la classification supervisée.

Nous avons aussi vu que les différents algorithmes (K-means, Random forest...) ont des performances différentes sur un même dataset donné, et leurs utilisation en petite et grande dimension. Dans le cas de la grande dimension, on retiendra que la méthode de l'ACP est très efficace pour supprimer des dimensions sans grand intérêt sur la variable à expliquer.

En somme, ce projet peut être vu comme un manuel sur la classification, qui donnerait à son lecteur toutes les notions théoriques pour comprendre en profondeur ce qu'est la classification, le supervisé, et le non supervisé. Quelques cas pratiques en petite et grande dimension permettront d'illustrer la théorie.

6 Annexes :

6.1 Code Non-Supervisée :

6.1.1 K-moyennes :

```
kmeans_groupe1<-function(x,v){
  #Permet de trancher sur le nombre de groupes à chosir
  #Prend un jeu de données en argument et le nombre de fois où Kmeans choisira
  #les points de départ au hasard
  #Renvoie la courbe d'inertie expliquée en fonction du nombre de groupes
  inertie.expl<-rep(0,times=10)
  for (k in 2:15){  #nombre minimum de groupes : nombre maximum de groupes
    clus<-kmeans(x,centers=k,nstart=v)
    inertie.expl[k] <-clus$betweenss/clus$totss
  }
  return(plot(1:15,inertie.expl,type="b",xlab="Nb de groupes",ylab="% inertie expliquée"))
}
```

```
kmeans_groupe2<-function(x){
  #Permet de trancher sur le nombre de groupe à chosir
  #Prends un jeu de données en argument
  #Renvoie la courbe de l'indice d'Harabaz en fonction du nombre de groupes
  test<-kmeansruns(x,krange=2:10,criterion="ch")
  return(plot(1:10,test$crit,type="b",xlab="Nb. de groupes",ylab="Silhouette"))
}
```

```
classif_kmeans<-function(x,u,v){
  #Etabli une classification via la méthode des K-means
  #Variables d'entrées : Un jeu de données, le nombre de groupe désiré, le nombre de fois
  #où Kmeans choisira les points de départ au hasard
  #Variable de sortie : l'objet "test" renferme toutes les propriétés de la classification
  test<-kmeans(x, centers =u, nstart = v)
  return(test)
}
```

Exemple sur les données de cancer :

```
temp<-HCPC(prostate)

w=which(temp$data.clust$clust==1) # Indice ligne groupe 1
v=which(temp$data.clust$clust==2) # Indice ligne groupe 2
u=which(temp$data.clust$clust==3) # Indice ligne groupe 3

sum(prostate$y[u]>0)/length(u) # 56\%
sum(prostate$y[v]>0)/length(v) # 53\%
sum(prostate$y[w]>0)/length(w) # 26\%
```

```r
test<-kmeans(prostate, centers =3, nstart = 3)

u=which(test$cluster==1) # Ligne groupe 1
v=which(test$cluster==2) # Ligne groupe 2
w=which(test$cluster==3) # Ligne groupe 3

mean(prostate$y[u]>0) # 0.53
mean(prostate$y[v]>0) # 0.26
mean(prostate$y[w]>0) # 0.56
```

```r
test<-kmeans(prostate, centers =2, nstart = 3)

u=which(test$cluster==1) # Ligne groupe 1
v=which(test$cluster==2) # Ligne groupe 2

mean(prostate$y[u]>0) # 0.57
mean(prostate$y[v]>0) # 0.40
```

Exemple sur les Iris de Fisher :

```r
test<-kmeans(fleur[,-5], centers =3, nstart = 3)

w=which(test$cluster==1) # Indice ligne groupe 1
v=which(test$cluster==2) # Indice ligne groupe 2
u=which(test$cluster==3) # Indice ligne groupe 3

sum(fleur[u,5]=="virginica")/length(u)  # 94.8%
sum(fleur[u,5]=="versicolor")/length(u)  #5.2%
sum(fleur[u,5]=="setosa")/length(u)  # 0% de bons classements

sum(fleur[v,5]=="versicolor")/length(v) # 77%
sum(fleur[v,5]=="virginica")/length(v) # 23%
sum(fleur[v,5]=="setosa")/length(v) # 0%

sum(fleur[w,5]=="setosa")/length(w) # 100%
sum(fleur[w,5]=="virginica")/length(w) # 0%
sum(fleur[w,5]=="versicolor")/length(w) # 0%
```

7 Code E.M. :

```r
library(MASS)
install.packages("mclust")
library(mclust)

library(HDclassif)
library("spls")
library(ade4)
library(FactoMineR)
library(fastcluster)

data("prostate")

data("iris")

######################data =iris ####################
data =iris

class <- iris$class
table(class)

reg=glm(iris$Sepal.Length~iris$Petal.Length,data=iris)
S=predict(reg,type="response")
X=iris$Sepal.Length
Y=iris$Petal.Length
plot(4:8,1:7,xlab="False Positive Rate",
     ylab="True Positive Rate",cex=.5)
for(s in seq(0,1,by=.01)){
  Ps=(S>s)*1
  FP=sum((Ps==1)*(Y==0))/sum(Y==0)
  TP=sum((Ps==1)*(Y==1))/sum(Y==1)
  points(FP,TP,cex=.5,col="red")
}

names(iris)
View(iris)
mc <- Mclust(iris, G=9) # 3 clusters
mc
plot(mc)

kmeans(iris, centers, iter.max = 10, nstart = 1,
       algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
                     "MacQueen"), trace=FALSE)

plot(mc, what=c("classification"), dimens=c(1,3))
# using 1st and 3rd column of the iris dataset
plot(mc, what=c("classification"), dimens=c(2,4))

plot(mc, what=c("classification"), dimens=c(1,4))


```

```r

# make dataset with points from two different multivariate normal densities
mydata <- rbind(mvrnorm(50, c(1,1), matrix(c(.5^2,.20,.20,.5^2), ncol=2)),
                mvrnorm(50, c(4,4), matrix(c(1,0,0,1), ncol=2)))
plot(mydata, pch=19)


mixclust <- Mclust(mydata)  # if G is not given, it will test between 1 to 9
plot(mixclust, what=c("classification"))  # plot the distinct clusters found



mc <- Mclust(iris[,1:4], G=3) # 3 clusters
plot(mc, what=c("classification"), dimens=c(1,3)) # using 1st and 3rd column of the iris dataset
# make dataset with points from two different multivariate normal densities
mydata <- rbind(mvrnorm(50, c(1,1), matrix(c(.5^2,.20,.20,.5^2), ncol=2)),
                mvrnorm(50, c(4,4), matrix(c(1,0,0,1), ncol=2)))
plot(mydata, pch=19)
mixclust <- Mclust(mydata)  # if G is not given, it will test between 1 to 9
plot(mixclust, what=c("classification"), main =" Algorithme E.M. sur les Iris de Fisher")  # plot the
densityMclust(mydata)
plot(densityMclust(mydata))
summary(mixclust)
mixclust
mstepE(names(data), datab, prior = NULL, warn = NULL)
length(names(data))
dim(data)
datab=data[2:150,1:4]
Z=as.matrix(datab)
z=t(Z)
mstep(iris)
help()
###############4 data=cancer ############################

data("prostate")
dim(data)
dim(iris)
data= prostate$x

data2=data[,1:5]
View(data)
mc <- Mclust(data2) # 3 clusters
mc
plot(mc)
library(ade4)
acp <- dudi.pca(prostate, scannf = FALSE)
names(acp)
data3=(pve <- 100*acp$eig/sum(acp$eig))
mc2 <- Mclust(data3)
mc2
summary(mc2)
plot(mc2, what=c("classification"), main =" Algorithme E.M. sur les Iris de Fisher" )
plot(densityMclust(data3))
```

```
###### data =aco=idity #####
data("acidity")
data =acidity
length(acidity)
mc5=Mclust(acidity)

plot(mc5)
```

7.1 Code Supervisé

7.1.1 Leave one out

Mise en place de la procédure leave one out pour la régression logistique.

```
import pandas as pd
from sklearn.datasets import load_iris

df = pd.DataFrame(load_iris()['data'],columns=load_iris()['feature_names'])
df['target'] = load_iris()['target']


#Régression logistique ne peut prédire que des résultats de la forme :



df = pd.get_dummies(df,columns=['target'])

setosa=df['target_0'];versicolor=df['target_1'];virginica=df['target_2']
del df['target_0'];del df['target_1'];del df['target_2']

l = LogisticRegression()
l.fit(df,setosa)
df['target_0']=l.predict_proba(df)[:,1]
l = LogisticRegression()
l.fit(df,versicolor)
df['target_1']=l.predict_proba(df)[:,1]
l = LogisticRegression()
l.fit(df,virginica)
df['target_2']=l.predict_proba(df)[:,1]


targets = pd.DataFrame(df,columns=['target_0','target_1','target_2'])
targets['best']=targets.max(axis=1)
target = []
for i in df.index:
    if targets.best[i]==targets.target_0[i]:
        target.append(0)
    if targets.best[i]==targets.target_1[i]:
        target.append(1)
    if targets.best[i]==targets.target_2[i]:
        target.append(2)

```

```
## Resultats :
sum(load_iris()['target']==target)
```

7.1.2 Procédure de validation croisée

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import auc

auc_ = []
accuracy = []
confusion = []
for i in range(5):

    train,test=train_test_split(df,train_size=0.7,test_size=0.3)
    y_train = train['y']
    y_test = test['y']
    del train['y'];del test['y']
    rf = RandomForestClassifier(n_estimators=300,max_depth=40)
    rf.fit(train,y_train)
    Y = rf.predict(test)
    Y_proba = rf.predict_proba(test)#[:,1]
    print('accuracy :')
    print(sum(Y==y_test)/len(Y))
    print('auc :')
    print(auc(y_test,Y_proba,reorder=True))
    auc_.append(auc(y_test,Y_proba,reorder=True))
    accuracy.append(sum(Y==y_test)/len(Y))
    confusion.append(confusion_matrix(y_test,Y))
```

7.1.3 Régression Logistique

```
import pandas as pd
pd.set_option('display.max_columns',100)

df = pd.read_csv('dataset_cancer.csv')
df.shape

#(102, 6034)

#On se retrouve ici avec un probleme a grande dimension, 6034 colonnes
#Validation croisée

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import auc

for i in range(5):
    train,test=train_test_split(df,train_size=0.7,test_size=0.3)
```

```
    y_train = train['y']
    y_test = test['y']
    del train['y'];del test['y']
    rf = RandomForestClassifier(n_estimators=1000,max_depth=40)
    rf.fit(train,y_train)
    Y = rf.predict(test)
    Y_proba = rf.predict_proba(test)[:,1]
    print('accuracy :')
    print(sum(Y==y_test)/len(Y))
    print('auc :')
    print(auc(y_test,Y_proba,reorder=True))
```

7.1.4 ACP

```
from sklearn.decomposition import PCA
p = PCA(n_components=100)
p.fit(df)
df = pd.DataFrame(p.transform(df))
l = []
nn=0
for i in p.explained_variance_ratio_:
    nn+=i
    l.append(nn)

plt.figure(figsize=(12,8))
plt.title('Pourcentage de variance en \n fonction du nombre d\'axes de l\'ACP')
plt.xlabel('Nombre d\'axes')
plt.ylabel('Pourcentage de la variance conservée')

plt.plot(l)
```

7.1.5 Forêts Aléatoires

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=8)
y = df['target']
del df['target']
rf.fit(df,y)
sum(load_iris()['target']==rf.predict(df))
```

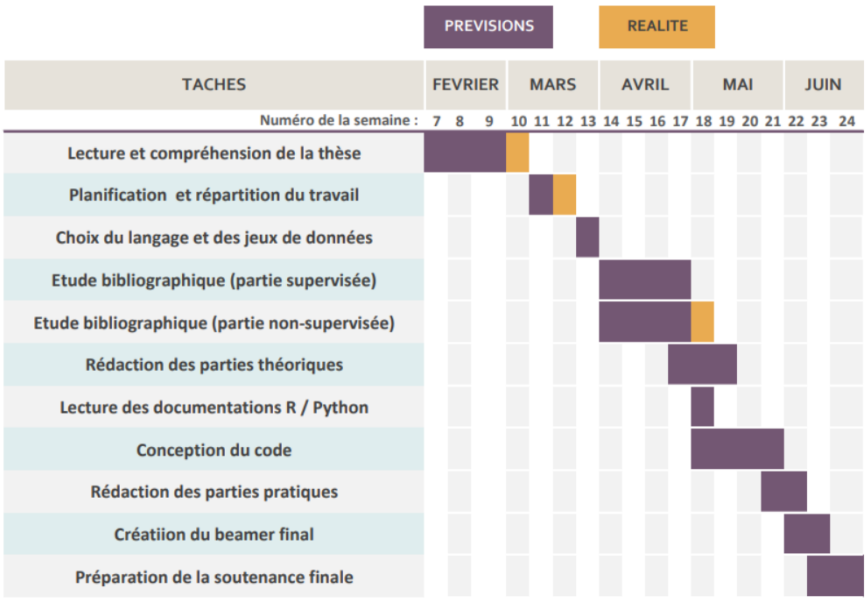
7.1.6 Visualisation de la matrice de confusion

```
import itertools
import numpy as np
import matplotlib.pyplot as plt

```

```
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

8 Diagramme de Gantt

FIGURE 20 – Diagramme de Gantt



— ABOUABDALLAH : Algorithme EM et Classification non supervisée
— FUCCELLARO : Classification supervisée