Projet RODM - Classification associative

(à effectuer seul ou en binôme, langage de programmation libre, Julia recommandé)

Nous considérons une liste de 189 patients.
Comme représenté en Table 1, pour chaque patient, nous connaissons :

— son âge ;

— 23 données médicales (taux d'albuminurie, taux de sodium, ...) ;

— s'il est malade ou non (classe du patient).

	caractéristiques			classe
Âge	**Albuminurie**	**Sodium**	...	**Malade ?**
48	4	0	...	notckd
⋮	⋮	⋮	⋮	⋮
57	0	0	...	ckd

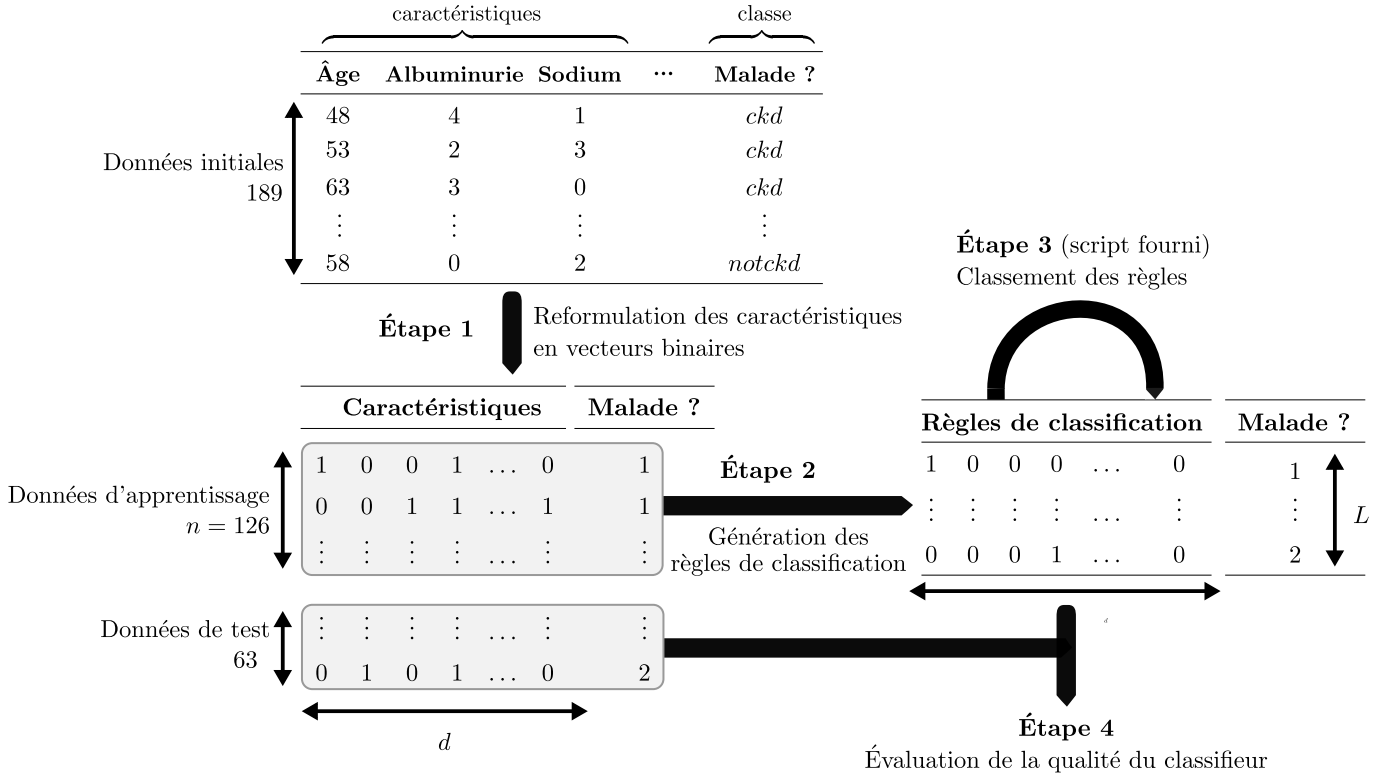
TABLE 1 – *Extraît des données initiales. Chaque ligne correspond à un patient. La dernière colonne contient la valeur ckd, s'il est malade.*

L'objectif de ce projet est de mettre en œuvre la technique de classification associative de Bertsimas, Chang et Rudin [1], présentée en cours, afin de diagnostiquer si les patients sont atteint d'une maladie reinale.

Le travail minimum demandé pour ce projet est :

— la réalisation des 4 étapes représentées dans le schéma ci-dessous ;

— le traitement d'au moins une des questions d'ouvertures de l'étape 5 (totalement ou partiellement suivant la difficulté).



Le code fournit avec ce sujet facilite l'implantation en julia. Il contient deux fichiers :

— `main.jl` : contenant l'inclusion des packages et l'appels aux principales fonctions (binarisation, création des règles, tri des règles et calculs des performances) ;

— `function.jl` : contenant le code des différentes fonctions dont une partie est à compléter.

Afin d'exécuter le programme il faut taper se déplacer dans le répertoire `src` et taper la commande suivante :

```
julia main.jl
```

Étape 1 Reformulation des caractéristiques sous la forme de vecteurs binaires (méthode `createFeatures`)

La méthode de classification considérée nécessite que chaque patient soit représenté par un vecteur de caractéristiques binaires. Cependant, la plupart des caractéristiques ne le sont pas.

1. Proposer une représentation binaire pertinente des caractéristiques.

   Remarque : Vous n'êtes pas obligé d'utiliser toutes les caractéristiques et plus vous aurez de caractéristiques, plus la résolution des problèmes d'optimisation risque d'être longue ou non optimale.
2. Reformuler les données initiales, situées dans le fichier "data/kidney.csv, en utilisant cette représentation. Pour cela vous pourrez vous aider de la méthode `createColumns` qui permet de binariser une colonne numérique à partir d'intervalles.
3. Créer le fichier de données "data/kidney_train.csv" contenant deux tiers des patients binarisés, choisi aléatoirement, ainsi qu'un fichier "data/kidney_test.csv" contenant le reste des patients binarisés.

Étape 2 Génération des règles de classification (méthode `createRules`)

Implanter l'algorithme vu en cours pour obtenir l'ensemble des règles de classification dans le fichier "res/kidney_rules.csv. Pour cela, vous pourrez compléter la méthode `createRules` du fichier functions.jl où les valeurs des différents paramètres de la méthode sont fixés.

Étape 3 Classement des règles (méthode `sortRules`)

Utiliser la méthode `sortRules` du fichier functions.jl afin de déterminer un classement optimal des règles. Après exécution, les règles ordonnées se trouveront dans le fichier "res/kidney_ordered_rules.csv".

Remarque : Par défaut le temps de résolution est limité à 300 secondes mais vous pouvez l'augmenter afin d'améliorer les performances (ou si aucune solution entière n'est trouvée).

Étape 4 Évaluation de la qualité du classifieur (méthode `showStatistics`)

Calculer la précision et le rappel du classifieur sur les données d'apprentissage et de test. Ces résultats vous semblent-ils pertinents ?

Étape 5 Questions d'ouverture

1. Comparer les performances du classifieur lorsque d'autres vecteurs de caractéristiques sont considérés.
2. Appliquer la méthode à un autre corpus (par exemple un de ceux figurant au lien suivant) ;
3. L'étape 2 consiste à générer les règles qui maximisent le support tout en minimisant la couverture. L'algorithme `RuleGen` fais le choix de fixer une borne supérieure sur la couverture afin de ne pas se trouver face à un problème bi-objectif. Dans cette question nous allons résoudre directement le problème bi-objectif. Pour ce faire, utilser les packages vOptGeneric et MultiJuMP de Julia qui permettent de résoudre facilement des problèmes de ce type en retournant une liste de solutions (chaque solution correspondant, ici, à une règle de classification). Étudier l'intérêt de cette approche par rapport à `RuleGen` (nombre de règles obtenues, temps de calcul, performance du classifieur, ...).
4. Afin d'accélérer la résolution, résoudre uniquement les relaxation linéaires des problèmes d'optimisation de l'étape 2 et/ou 3 (*i.e.*, résolution de problèmes continus au lieu de problèmes entiers). Vous arrondirez les solutions obtenues et comparerez les performances. Évaluer ensuite dans quelle mesure cela vous permet d'augmenter le nombre de caractéristiques considérées ou de règles générées, puis comparez une nouvelle fois les performances.

5. Est-ce que cette approche peut s'adapter à la classification à plus de deux classes ? Justifier et le tester le cas échéant.
6. Certaines des inégalités du problème de l'étape 3 ont pour principal objectif de renforcer la relaxation linéaire de la formulation. Déterminer celles qui sont les plus utiles (si elles s'avèrent effectivement utiles).
7. Évaluer l'impact de la limite de temps de résolution de l'étape 3 sur le gap et sur la qualité du classifieur obtenu.
8. Tout autre contribution qui peut vous sembler pertinente du point de vue des performances ou de l'application peut également faire l'objet d'une question d'ouverture.

Étape 6 Rendu

A l'issue de ce projet (data limite de rendu le 11/03), vous rendrez vos fichiers sources, les fichiers de données générés ainsi qu'un rapport qui contiendra notamment :

— Une description argumentée de la représentation binaire que vous avez choisie ;
— Le nombre de règles générées, le temps de calcul associé, le solveur choisi, ainsi que sa version, et les caractéristiques de la machine utilisée (type et nombre de processeurs et mémoire RAM) ;
— Le nombre de règles du classifieur et le temps de calcul pour les ordonner ;
— Les performances de votre classifieur (précision et rappel).
— Un tableau représentant les règles ordonnées de votre classifieurs ainsi que leur classe accompagnés d'une description de la façon dont le classifieur fait ses choix.
— Le travail effectué sur les questions d'ouvertures.
— La liste des opérations à effectuer pour reproduire vos expériences (quels programmes exécuter, avec quels paramètres, dans quel ordre, ...).

Références

[1] Allison Chang, Dimitris Bertsimas, and Cynthia Rudin. An integer optimization approach to associative classification. In *Advances in neural information processing systems*, pages 269–277, 2012.