Projet RODM - Classification associative

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

Nous considérons une liste de 887 passagers du Titanic.
Comme représenté en Table 1, pour chaque patient, nous connaissons :

— son sex
— la catégorie de son billet (classe 1, 2 ou 3)
— son âge
— le nombre de parents qu'il avait à bord du Titanic
— le prix de son billet
— s'il a survécu au naufrage (classe du passager)

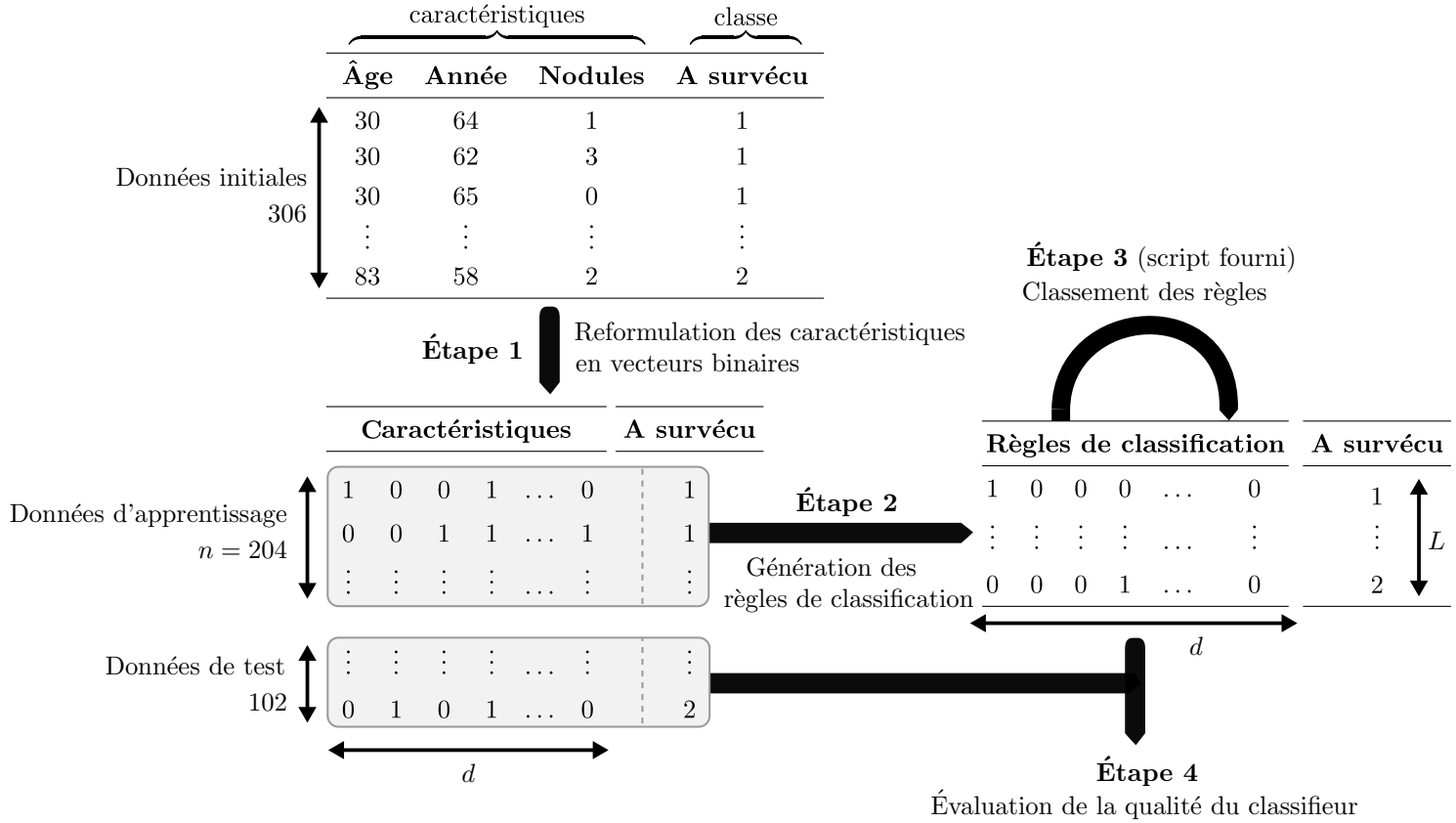
	caractéristiques				classe
Catégorie	**Sex**	**Âge**	**Parents**	**Prix**	**A survécu**
3	M	1	22	7.25	0
1	F	38	1	71.28	1
⋮	⋮	⋮	⋮	⋮	⋮
3	M	32	0	7.75	0

TABLE 1 – *Extraît des données initiales. Chaque ligne correspond à un passager. La dernière colonne contient la valeur 1, si le passager a survécu.*

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 prédire les chances de survie des patients.

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).



Étape 1 Reformulation des caractéristiques sous la forme de vecteurs binaires

La méthode 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.
2. Reformuler les données initiales, situées dans le fichier "data/titanic.csv, en utilisant cette représentation (pour cela vous pourrez compléter la méthode `createFeatures` du fichier functions.jl). Pour cela, vous pourrez vous aider de la méthode `createColumns` qui permet de lire une colonne d'une table et de la binariser suivant des intervalles.
3. Créer le fichier de données "data/titanic_train.csv" contenant deux tiers des patients reformulés, choisi aléatoirement, ainsi qu'un fichier "data/titanic_test.csv" contenant le reste des patients reformulés.

Étape 2 Génération des règles de classification

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

Étape 3 Classement des règles

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

Remarque : Par défaut le temps de résolution est limité à 600 secondes mais vous pouvez l'augmenter afin d'améliorer les performances en éditant la ligne : `m = Model(solver=CplexSolver(CPX_PARAM_TILIM=600))`

Étape 4 Évaluation de la qualité du classifieur

1. Calculer la précision du classifieur obtenu en l'utilisant sur les patients figurant dans le fichier "data/titanic_test.csv"
2. Calculer le rappel du classifieur en utilisant le classifieur sur le fichier "data/titanic_train.csv".

Étape 5 Questions d'ouverture

1. Comparer les performances du classifieur lorsque d'autres vecteurs de caractéristiques sont considérés.
2. Tester l'influence des divers paramètres de la méthode.
3. Appliquer la méthode à un autre corpus (par exemple un de ceux mentionnés en Section 5 de [1]) ;
4. Les données d'apprentissage, que vous avez utilisées, sont constituées de deux tiers des patients (notés tiers A et tiers B) tandis que le tiers restant (noté tiers C) constitue les données de test. Afin de limiter les risques de biais dans les résultats, calculer les performances des deux classifieurs obtenues lorsque l'apprentissage est effectué en utilisant les tiers A et C, puis B et C.
5. 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, ...).

Étape 6 Rendu

A l'issue de ce projet (data limite de rendu le 10/03), 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, rappel).
— Un tableau représentant les règles ordonnées de votre classifieurs ainsi que leur classe.
— 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.