Projet Apprentissage artificiel
Présentation des choix d'implémentation

Jonathan Deramaix
Jules Leguy

Sources et instructions d'utilisation : `https://github.com/juleguy/DecisionTree`

1 Données

1.1 Extraction des données

Le programme utilise les *DataFrames* de la librairie Pandas pour stocker les données, et utilise la librairie arff2pandas pour convertir les fichiers arff en tableaux Pandas. L'intérêt des *DataFrames* de Pandas est qu'elles permettent d'itérer facilement sur les colonnes et les lignes, à partir des index ou des noms.

1.2 Représentation des données

1.2.1 Tableaux de données

Une fois les données chargées dans une première *DataFrame* pandas, on extrait les valeurs nominales possibles pour chaque attribut (arff2pandas les concatène au nom de la colonne du tableau) et on stocke ces valeurs dans un dictionnaire ayant les attributs comme clés et la liste des valeurs possibles comme valeurs.

À partir du tableau initial, on créé deux tableaux contenant les exemples pour chaque classe (positive ou négative), qu'utilisera ensuite l'algorithme de couverture séquentielle pour générer les règles.

Si le programme est en mode prédiction, on créé également deux tableaux composant le jeu de test. Ces tableaux sont alors créés en utilisant la bibliothèque Scikit-Learn, qui permet de séparer facilement le jeu de données en un jeu d'entraînement (80% des données) et un jeu de test (20% des données). Cette séparation est aléatoire mais utilise une graine aléatoire fixe afin que les tableaux soient toujours identiques pour une entrée donnée.

1.2.2 Logique

Afin de représenter les données liées à la logique (les couples attribut-valeur et les règles), nous avons créé deux classes dédiées. En dehors du simple affichage des données, la classe représentant une règle possède une méthode permettant de calculer si une ligne d'un tableau est conforme à la règle.

2 Architecture et algorithmes

2.1 Architecture générale

Le script python qu'exécute l'utilisateur se charge de la gestion des arguments fournis par l'utilisateur. Il instancie un objet *FoilProp* à qui il transmet et les arguments et qui se charge de toute la logique du programme.

2.2 FoilProp

2.2.1 Méthodes

Il s'agit de la classe principale du programme. Son fonctionnement est inspiré des modèles de la bibliothèque Scikit-Learn. Elle contient en effet une méthode *fit(filename)* qui va créer un ensemble de règles en appliquant l'algorithme «FOIL Propositionnel» sur les données représentées par le fichier dont le nom est passé en paramètres.

Elle contient également une méthode *predict()* qui va tester les règles générées sur tous les exemples du jeu de test, et qui va compter le nombre de vrais positifs, vrais négatifs, faux positifs et faux négatifs. Ces quantités permettront par la suite de calculer les scores de précision, rappel et f1.

2.2.2 Cas particuliers du calcul du gain

À chaque itération de la boucle interne de l'algorithme de création de règles, le choix du meilleur littéral pour former une prémisse de la règle s'effectue en utilisant la fonction de calcul du gain. Le littéral sélectionné est le littéral possédant le meilleur gain. Mais le calcul du gain faisant intervenir des divisions, il peut dans certains cas ne pas être possible. Lorsque son calcul n'est pas possible, notre algorithme renvoie la valeur *None*. Et si le calcul du gain n'est possible sur aucun littéral, il renvoie un littéral aléatoire parmi la liste des littéraux possibles.

Un autre cas particulier survient quand l'union de l'ensemble d'exemples positifs forme le même ensemble que l'union des exemples de l'ensemble d'exemples négatifs (comme dans le jeu *mushroom_train* au bout d'un certain nombre d'itérations). Dans ce cas, le gain de tous les littéraux est nul, et si l'on renvoie un littéral qui est présent dans tous les exemples, alors aucune ligne n'est éliminée et la boucle interne de l'algorithme de génération des règles boucle à l'infini.

Pour pallier ce problème, lorsque tous les gains sont nuls, l'algorithme applique une heuristique cherchant le couple d'exemples (ex_pos, ex_neg) ayant le plus de similarité (le plus de littéraux en commun), tel que ex_pos appartient à l'ensemble d'exemple positifs et ex_neg appartient à l'ensemble d'exemples négatifs. Une fois le meilleur couple trouvé, il renvoie le premier littéral différant entre les deux exemples du couple.

2.2.3 Attributs et propriétés

La classe *FoilProp* contient en outre un certain nombre d'attributs et de propriétés. Les données nécessaires pour l'application des algorithmes de création de règles et de prédiction sont stockées en tant qu'attributs privés. Les données accessibles uniquement en lecture ou ayant besoin d'un traitement avant d'être envoyées (la taille des jeux de données, les calculs d'efficacité de la prédiction) le sont via des propriétés.

2.3 Gestion des valeurs manquantes

Certains jeux de données contiennent des valeurs manquantes qui ne sont pas prévues par l'algorithme qui génère les règles. Le comportement par défaut du programme est d'ignorer les exemples contenant des valeurs manquantes. D'autres comportements sont cependant disponibles. On peut choisir de remplacer les valeurs manquantes par une des valeurs possibles choisie aléatoirement, ou de remplacer les valeurs manquantes par la valeur de l'attribut ayant le plus d'occurrences pour la classe de l'exemple.

Le programme utilise le design pattern Stratégie [1] pour encapsuler la stratégie de gestion des valeurs manquantes dans différentes classes. La classe correspondant à la stratégie spécifiée par l'utilisateur est instanciée au début du programme puis se charge de remplacer les valeurs manquantes ou de les ignorer.

1. `https://sourcemaking.com/design_patterns/strategy`

3 Utilisation

Les commandes utilisables sont décrites dans le fichier README.md disponible sur la page github du programme.