Projet Back2Back testing

Juliette CHOUDJAYE
François MATVIENKO
Armel MOTH
Lydia VIJAYARAJAH

15 mars 2018

1 Introduction

Le but de notre projet était de permettre à un utilisateur de comparer les performances (grâce à l'accuracy) de trois libraires (Weka, Renjin, SparkML) de machine learning.

Pour faire cela, nous avons tout d'abord réalisé une classe par librairie, permettant d'éxecuter les différentes méthodes disponibles dans les librairies. Par ailleurs, pour que notre application soit simple à utiliser, nous avons implémenter des «outils» pour faciliter l'utilisation.

2 Les classes

Nous avons décidé d'utiliser le *design pattern* "Adapter". En effet, cela nous a permis de coder librement les classes de chaque librairie (*SparkML*, *Renjin* et *Weka*) pour ensuite pouvoir les mettre au format de la classe abstraite *Library* par le biais de classes adapter (*SparkMLLib*, *RenjinLib* et *WekaLib*).

Classe Library :
C'est une classe abstraite dont héritent les autres librairies, afin de définir des méthodes communes et de faciliter la comparaison (classe adaptor du design pattern).

Classes SparkML, Renjin et Weka :
Ce sont les classes dans lesquelles nous implémentons les différentes méthodes disponibles librement (classes adaptee du design pattern adapter).

Classes SparkMLLib, RenjinLib et WekaLib :
Ce sont les classes qui permettent de passer des classes précédentes au format de la classe Library (classes concrete adapter du design pattern).

Classe Split CSV :
Cette classe nous permet de diviser en deux un csv, afin que l'échantillon de test et celui d'apprentissage soit le même pour les deux librairies à comparer.

Classe Comparateur :
C'est la classe qui prend deux librairies en arguments pour pouvoir comparer leurs performances.

Classe Lanceur :
C'est dans cette classe que se trouve la main qui permet de lancer notre application.

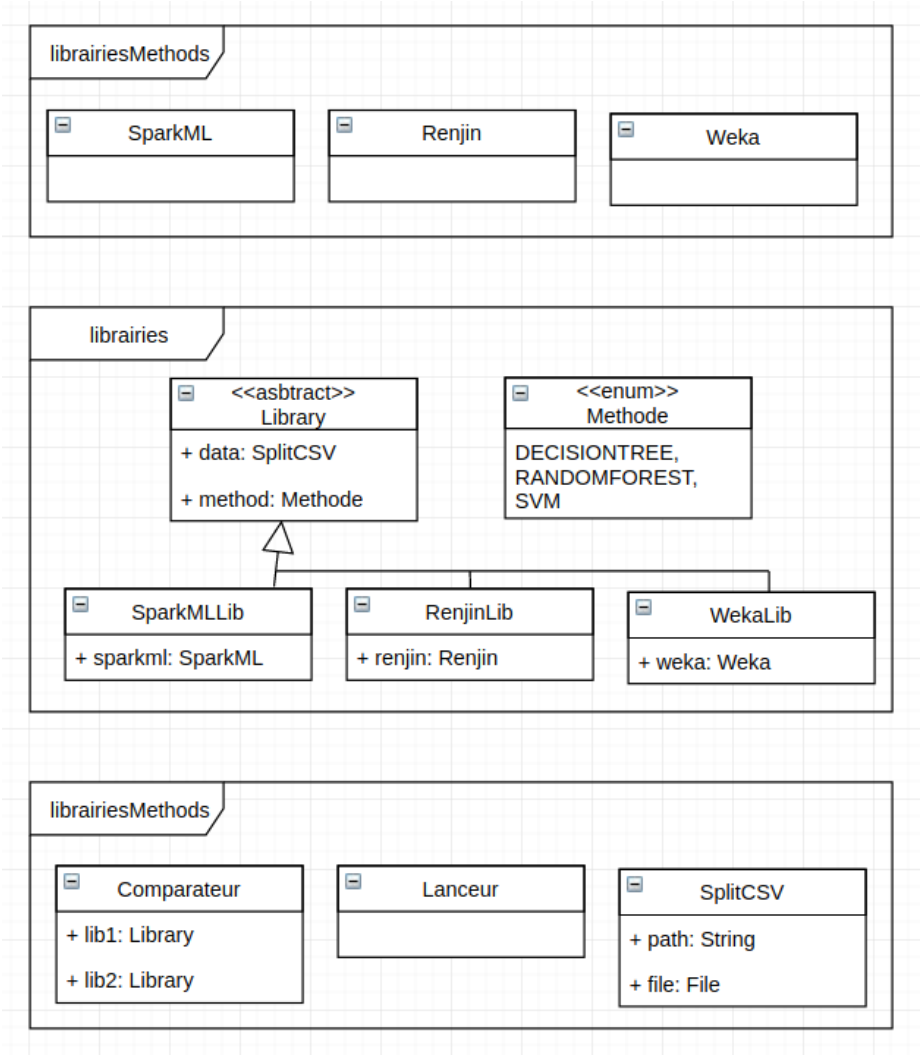

FIGURE 1 – Diagramme de classes

3 Les librairies

3.1 SparkML

La librairie SparkML a été assez difficle à utiliser, notamment car nous voulions nous fonder sur la structure de données RDD qui est la particularité de cette librairie. Néanmoins nous sommes arrivés à ce que nous voulions à savoir : être capable de prendre quasiment n'importe quel type de dataset et d'effectuer sur ce dernier des méthodes de classification. Aujourd'hui la librairie sparkML permet de calculer des taux d'accuracy pour les méthode Random Forest, DecisionTree et SVM.

La façon global dont fonctionne la librairie est la suivante : On crée un objet de la classe SparkMLLib avec en argument un splitCSV (un chemin vers un jeux de donnée de test et un autre de train), une méthode, et facultativement un dictionnaire pour les paramétres de la méthode.
L'objet va réécrire deux nouveaux .csv au format compatible avec les méthodes de sparkML (par exemple pas de string, remplacer les modalités de variable qualitative par 1.0,2.0,...) Une fois que cela est fait l'objet va vérifier que la méthode appelée est disponible pour la librairie sparkML (il regarde dans la liste static si la méthode est présente) .
Si la méthode est disponible, il définit alors tous les paramétres pour cette méthode à leur valeur par défaut, puis si dans la hashmap passée en paramétre au début de la classe, il trouve des paramétres disponibles pour cette méthode il remplace leur valeur par défaut par la valeurs associée dans la HashMap. Ainsi si l'utilisateur passe des paramétres non disponible cela n'impacte pas le programme mais on ne gere pas le cas ou des valeurs abérantes sont données à des paramétres effectivement disponible.
Une fois ce travail affectué, la classe SparkMLLib appel simplement la méthode de l'objet AlgoSparkML associé à la méthode statistique choisi avec les paramétres définit plus haut.

3.2 Weka

La librairie Weka a été très intéressante à implémenter.

Tout d'abord elle dispose de ses propres méthodes pour lire les CSV, ce qui rend l'importation des données très efficace et rapide. En effet, grâce au CSV Loader, il est assez simple de donner l'échantillon d'apprentissage et celui de test.

Avec Weka, nous avons implémenté la méthode pour obtenir un J48 Tree, qui est un arbre de décision propre à Weka, et celle du Random Forest.

Au final, cette partie est celle qui nous a posé le moins de problèmes à réaliser.

De plus, cette librairie est très intéressante car bien que moins souple que les autres et offrant un peu moins de possibilités, elle est certainement la plus simple d'utilisation pour un novice ; c'est aussi celle qui s'intègre le plus facilement dans un code java car les méthodes sont relativement aisées à prendre à main et adapter à une situation.

3.3 Renjin

La librairie Renjin était plutôt facile à utiliser. En effet le lien entre Java et R semble ici être assez proche du lien entre R et C++. Ce sont des langages complémentaires, ainsi Java permet d'écrire les codes R grâce aux classes Renjin ScriptEngineFactory et ScriptEngine. Pour faire court, Java va surtout faire office de lecteur de script R.

Ainsi il s'agissait dans notre cas de choisir une modélisation qui nous permettrait de rendre l'utilisation de la libraires Renjin facile, tout en gardant la souplesse du code et surtout de l'application finale. Ceci nous a ammené à définir nos arguments Java (ou variables R) d'un côté, puis nos fonctions effectuant les arbres de décision ou des random forest d'un autre côté. Dans un appel au script R sur lequel nous travaillions, les arguments une fois initialisés sur Java étaint convertis et enregistrés en variables R. Puis ces variables servaient d'arguments aux appels de fonctions contenues dans le script. Le résultat de chacune de ces fonctions (accuracy) était ensuite converti et récupéré par Java.

D'autre part l'utilisation de la librairie Renjin nous a aussi obligé à vérifier que chaque package utilisé dans R existait bel et bien dans Renjin. Sinon les fonctions du dit package ne marcheront pas.