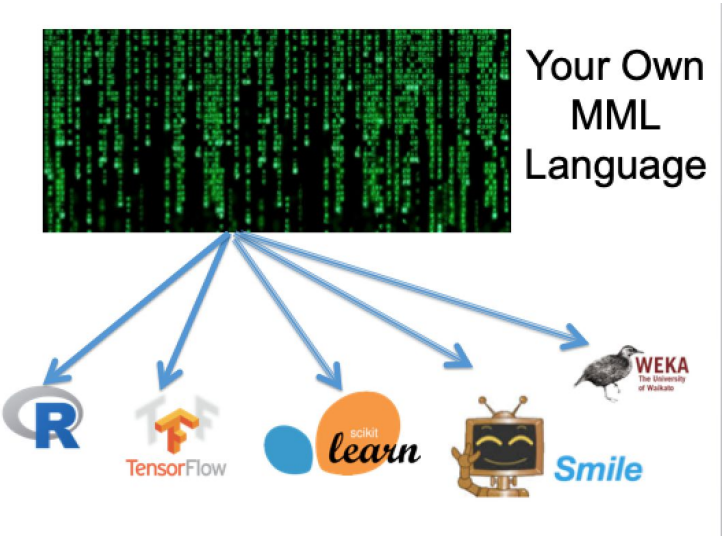
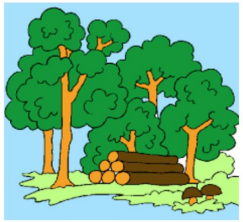
Java, JSON, Compilateurs, MML

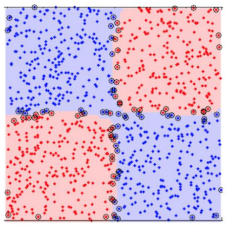
Q0: Former des groupes de 4 (ou 5 max)

On souhaite développer **Multi Machine Learning (MML)**, une solution écrite en Java qui permettra d'effectuer des tâches de machine learning (e.g., classification). L'idée est qu'un utilisateur configure sa tâche de machine learning (eg via un fichier JSON) puis des programmes exécutables, écrits en Python, R, Julia, sont générés automatiquement.
Le développement s'effectuera en plusieurs étapes, avec l'utilisation de Java et JSON
https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

Dans notre contexte, l'idée est de pouvoir lancer facilement des campagnes de comparaison entre des mises en œuvre distinctes d'algorithmes de Machine Learning (ML) et opérant sur des mêmes jeux de données. Ce mini-projet sera l'occasion de pratiquer les différents concepts et outils abordés en cours (tests, Git, Maven, Docker, etc.)







Multi Machine Learning Language (MML)

Le but du projet est de travailler sur un outil permettant d'automatiser des tâches de ML. En effet de très nombreuses implémentations sont maintenant disponibles, dans différents langages (Python, Java, Scala, R, etc.). Nous souhaiterions disposer d'un outil (MML) capable de simplifier l'exécution et la comparaison de résultats fournis par ces différentes implémentations.

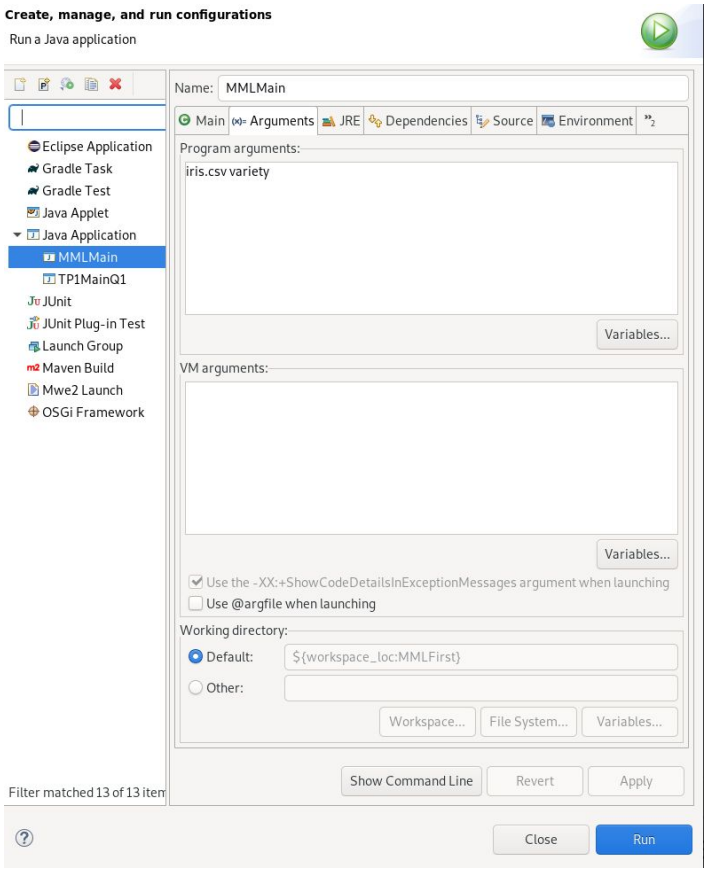
Nous commencerons en restreignant quelque peu le problème:

- On ne s'intéresse qu'aux algorithmes *supervisés* de *classification et encore plus précisément aux* ***decision tree*** avec une seule variable cible https://en.wikipedia.org/wiki/Statistical_classification.
- On prendra en entrée des données au format *CSV* (e.g., le fameux dataset Iris https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/datasets/iris.csv). Le CSV pourra avoir un séparateur autre que ", "
- On peut paramétrer la stratégie d'évaluation
    - Soit on découpe le jeu de données en deux (training/test) et dans ce cas le pourcentage de données utilisées pour l'apprentissage et le test peut être paramétré (par défaut, 70% training, 30% testing)
    - Soit on utilise la cross-validation (ici encore essayer de fournir les moyens de paramétrer la manière d'effectuer la cross-validation)
- On peut spécifier les variables prédictives et la variable cible
    - Par défaut, toutes les variables sont prédictives sauf la dernière colonne du CSV qui est la variable cible
- Il est possible de spécifier des valeurs d'hyper-paramètres
- En sortie du programme, on souhaite calculer la *précision et/ou le recall et/ou l'accuracy et/ou f1...*
- On "vise" trois implémentations:
    - En Python et en utilisant la librairie scikit-learn https://scikit-learn.org/stable/modules/tree.html
    - En R et en utilisant le package rpart https://cran.r-project.org/web/packages/rpart/rpart.pdf
    - En Julia et en utilisant la librairie DecisionTree https://github.com/bensadeghi/DecisionTree.jl

L'implémentation a été initiée ici:
https://github.com/acherm/multimachinelearning
avec le support de 2 implémentations (Python, R).
Le support est très incomplet (eg il est obligatoire de spécifier la variable cible, seul l'accuracy est supporté, etc.) et la solution a été très peu testée (eg sur différents dataset).

Installation

git clone https://github.com/acherm/multimachinelearning dans un terminal
import via Eclipse (as a Maven project)
exécution de MMLMain en spécifiant des arguments (run as => configurations)
exécution des tests unitaires



Amélioration du code

Pour la séance du 27/01/2021 (et 28/01/2021), nous allons travailler sur au moins deux axes.

Le premier concerne la configuration (ou paramétrisation) de MML.
Nous allons refactoriser le code pour le rendre ***configurable*** (création de classes “domaine”, utilisation de JSON, etc.)

refactoriser (= structurer) le code de telle sorte que generateCode ne prenne plus de paramètres en entrée

exercice: supprimer args[0] et args[1] dans le code actuel et le remplacer par un fichier JSON
puis prendre en compte les paramètres du fichier JSON lors de la génération de code (étendre ConfigurationML puis implémenter le nécessaire dans generateCode de la classe PythonMLExecutor et de la classe RLanguageMLExecutor)

Le deuxième concerne le “test” de l’implémentation.
Nous allons refactoriser le code pour le rendre ***testable*** (rendre observable code, poser des assertions, etc.)

Rendu

Date limite: 10/03/2021
Par groupe de 4 (ou 5 max.)

- un seul rendu par groupe (git, cf ci-dessous)
- “Rapport” à rendre de manière individuel

- Un repository “git” hébergé sur Github ou gitlab
- Instructions (via README.md) pour utiliser votre solution et répliquer vos résultats
- Implémentation des 3 compilateurs
    - dont le compilateur Julia
- Support du fichier de configuration (JSON)
    - pour lancer une tâche de machine learning, l’utilisateur n’aura qu’à spécifier un fichier JSON
    - concevoir une syntaxe et une organisation de l’information en JSON
    - écrire les classes pour lire le JSON et organiser l’information sous forme d’objets Java… les prendre ensuite en compte dans les différents compilateurs
- Diagramme de classes UML pour documenter le projet
- Support de Docker
    - On doit pouvoir exécuter les programmes générés dans un Docker

    - Fournir un Dockerfile
    - Ré-implémenter run() dans les trois compilateurs
- Maven
    - Packaging
    - Lancement des tests
- Tests unitaires
    - Pour démontrer que votre solution est robuste dans différents cas
- Qualité du code
- Etude comparative des trois compilateurs (rapport)