INGÉNIEUR EN SCIENCES INFORMATIQUES

RAPPORT FINAL D'APPRENTISSAGE

Conception, réalisation et intégration d'une plateforme de Meta-Learning[1]

2017-2018

Apprenti: Günther Jungbluth
Maître d'apprentissage: Sébastien Mosser
Tuteur enseignant: Diane Lingrand

Entreprise: CNRS, Laboratoire I3S, Equipe Sparks, Projet ROCKFlows

Résumé:

ROCKFlows est un projet visant à simplifier la création de workflows d'apprentissage automatique en proposant, pour un problème donné, des workflows idoines tirés d'un portfolio. Pour construire ce portfolio des expérimentations doivent être réalisées. Celles-ci sont utilisées pour construire des modèles de prédiction des performances des workflows pour un type de problème et un jeu de données. ROCKFlows possédait un système d'exécution d'expérimentations mais ce dernier devait évoluer pour répondre à de nouvelles contraintes tel que la prise en charge d'un grand nombre d'expérimentations, l'automatisation les

[1] https://en.wikipedia.org/wiki/Meta_learning_(computer_science)

différentes tâches afférentes à un portfolio dont la composition de workflows valides, bootstrapper le système lui-même, ...

Remerciements

Je tiens à remercier Mireille Blay et Sébastien Mosser pour leur accompagnement et leur aide pendant cette alternance.
Je remercie Diane Lingrand, Frédéric Precioso et Michel Riveill pour avoir répondu aux différentes questions en rapport avec les besoins des experts en apprentissage automatique ainsi que Philippe Collet pour avoir répondu à mes questions sur les *feature model*[2].
Je remercie également Benjamin Benni pour avoir proposé et discuté de solutions aux besoins de ROCKFlows pendant mon apprentissage ainsi que Clément Duffau pour les discussions que nous avons eu ensemble sur des nouvelles perspectives du projet.
Je remercie François Montigny pour son travail lors de son stage d'été sur ROCKFlows.
Et je remercie également tous les membres de l'équipe SPARKS du laboratoire I3S pour leur convivialité.

[2] https://en.wikipedia.org/wiki/Feature_model

Plan du rapport

1. Introduction

ROCKFlows est un projet du laboratoire I3S mené par Mireille Blay et Frédéric Précioso. Il vise à aider un utilisateur novice en apprentissage automatique à choisir une solution adaptée à son problème. Un problème d'apprentissage automatique est défini par une tâche, comme par exemple de la classification[3], et par un jeu de données. Pour cela, l'application repose sur un portfolio de *workflow*[4] d'apprentissage automatique. Un *workflow* est défini par une succession de prétraitements suivie d'un algorithme d'apprentissage automatique. Un prétraitement transforme un jeu de données en un autre jeu de données. On appellera chacun de ces composants un algorithme. La figure 1 présente un *workflow* d'apprentissage automatique.

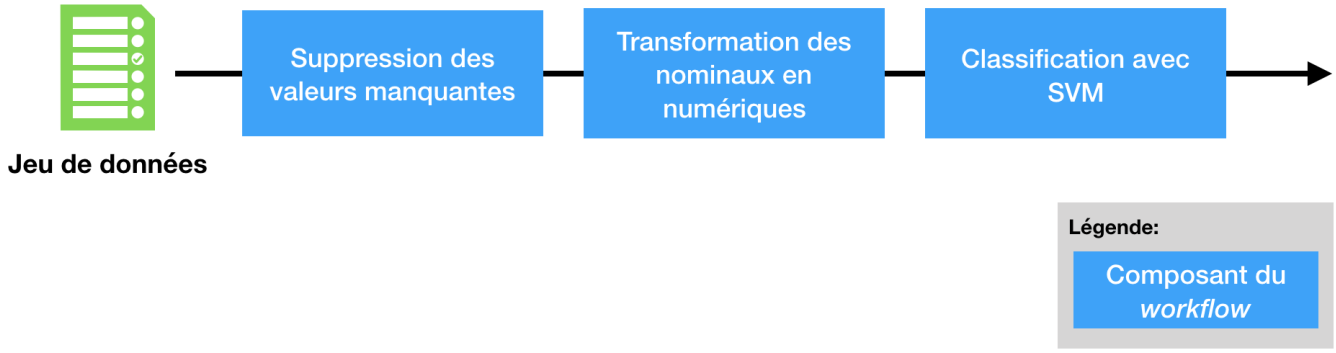


Figure 1: Un workflow d'apprentissage automatique.
SVM[5] (Support Vector Machine) est un algorithme de classification.
Ici, le jeu de données est transformé par deux traitements avant d'être classifié.

Prenons un exemple, un utilisateur ayant peu de connaissance en apprentissage automatique possède un jeu de données répertoriant les revenus quotidiens de son commerce en fonction des jours de la semaine, des jours fériés, des évènements et des conditions météorologiques. Cet utilisateur pense pouvoir prédire ses futurs revenus en fonction de ce jeu de donnée mais ne possède pas la connaissance requise pour choisir la bonne combinaison d'algorithmes d'apprentissage automatique. Il utilise alors la plate-forme web ROCKFlows pour obtenir un *workflow* correspondant à son problème ainsi que son code d'exécution. Il peut ensuite exécuter ce *workflow* et prédire son chiffre d'affaire du lendemain et des jours suivants. Cet utilisateur correspond à l'un des utilisateurs principal de ROCKFlows. Ce rapport présentera également un type d'utilisateur voulant utiliser de l'apprentissage automatique au coeur d'un projet informatique sans passer par une interface graphique.

Pour permettre ce type de recommandation de *workflows*, ROCKFlows utilise des méthodes d'apprentissage automatique qui se basent sur des exécutions de *workflows* déjà réalisés. On appelle ces exécutions des *expérimentations*. Ces différentes expérimentations sont utilisées au sein de ROCKFlows pour prédire les performances d'un portfolio de *workflows* d'apprentissage automatique.

[3] https://fr.wikipedia.org/wiki/Classification
[4] Un *workflow* d'apprentissage automatique est un enchaînement de prétraitements sur un jeu de données suivi d'un algorithme d'apprentissage automatique
[5] https://en.wikipedia.org/wiki/Support_vector_machine

ROCKFlows effectue des prédictions sur la performance des *workflows* d'apprentissage automatique. C'est ce qu'on appelle du méta-apprentissage automatique. Par exemple, ROCKFlows peut prédire la précision du *workflow* présenté en figure 1 pour un jeu de données fournit par un utilisateur.
Ce portfolio de *workflows* est défini par un ensemble d'algorithmes. Cet ensemble s'enrichit de jour en jour par la communauté de l'apprentissage automatique rendant le choix d'un *workflow* parmi le portfolio de plus en plus difficile. En effet, les experts de ce domaine développent de façon continue des nouveaux algorithmes dans des langages différents. C'est pour cela que ROCKFlows cherche à automatiser la sélection du *workflow* pour un problème donné tout en étant incrémental pour suivre l'évolution de l'apprentissage automatique.
Pour pouvoir lancer les expérimentations constituant la base d'apprentissage de ROCKFlows, un système contrôlant et automatisant ces exécutions doit être mis en place. C'est plus particulièrement sur ce point qu'a porté mon apprentissage même si j'ai dû pour cela aborder de nombreux autres aspects.

Ce rapport décrit le travail effectué lors de mon apprentissage à travers la mise en exergue de différents besoins. Le but de ce rapport n'est pas de décrire en détails les solutions apportées mais de montrer leur pertinence par rapport aux besoins auxquelles elles répondent. Ce choix est étayé par le contexte de recherche dans lequel ce projet a été réalisé, nous avons en effet dû faire face à de nombreux changements de spécifications pour répondre aux difficultés rencontrées. De plus, pour répondre aux besoins de ROCKFlows une approche plus ingénierie que programmation du projet a été utilisée à travers différentes phases d'analyses et de conception.

Ce rapport présente dans un premier temps les objectifs de mon apprentissage à travers une analyse de la plate-forme, puis (2.) le travail que j'ai réalisé à travers la conception, le développement et l'intégration de différents systèmes au sein de ROCKFlows (3.). La partie (4.) synthétise les principaux résultats obtenus lors de mon apprentissage, tandis que la partie (5.) expose les perspectives avant de conclure.

2. Analyse des objectifs de mon apprentissage

Dans ce qui suit je montre comment ces différents aspects ont été pris en charge pour répondre aux objectifs initiaux d'automatisation, validation, incrémentalité et passage à l'échelle. Il est important de noter que la spécification même des objectifs entre dans le cadre de mes contributions. En effet, pour atteindre ce degré de spécification, il a été nécessaire d'analyser la plateforme et le domaine de l'apprentissage automatique avant de les expliciter.
La figure 2 présente les différents aspects travaillés lors de mon apprentissage.

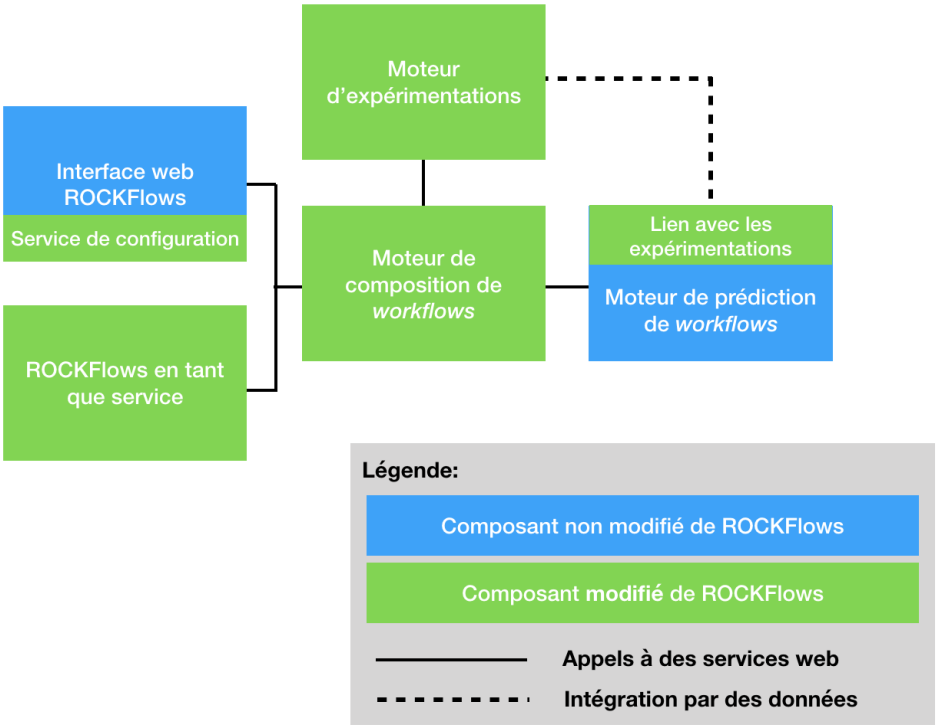


Figure 2: Architecture de haut niveau de ROCKFlows. En vert sont représentés les aspects travaillés lors de mon apprentissage.

2.1 Composition de *workflows*

L'une des grands problématiques de ROCKFlows est de générer un portfolio valide pour un problème donné. Ce portfolio doit être restreint aux *workflows* qui sont valides syntaxiquement et sémantiquement.

Un *workflow* est dit *valide syntaxiquement*[6] si les conditions d'exécution de chaque composant du *workflow* sont valides. Par exemple, le *workflow* "`TransformationNumériqueEnNominal; Normalisation; SVM`" n'est pas valide car le pré-traitement `Normalisation` exige des données numériques alors que l'algorithme précédent celui-ci supprime les données numériques.

Un *workflow* est dit *valide sémantiquement* si les conditions définies par un expert en apprentissage automatique sont respectées. Par exemple, le *workflow* "`Normalisation;`

[6] Nous choisissons ce terme, par analogie avec des règles grammaticales, les algorithmes définissant une sorte de langue

`SupressionValeurManquante; SVM`" est invalide car il ne respecte pas la contrainte : "La suppression des valeurs manquantes se fait toujours en premier".
Le but est donc ainsi de générer un portfolio de *workflows* en fonction des conditions syntaxique et sémantique et d'un jeu de données. Cette génération doit passer à l'échelle pour un grand nombre (centaine) de pré-traitements.

2.2 Conception et réalisation d'une plateforme d'expérimentation

Pour permettre l'ajout de nouveaux algorithmes écrits dans différents langages de programmation et ainsi enrichir la plate-forme d'algorithmes existant développés par la communauté de l'apprentissage automatique, le module d'expérimentations de ROCKFlows doit être changé.
Dans cette partie, j'analyse l'ancien système d'expérimentations (version 1) et je mets en avant les limites de celui-ci.

L'ancienne version du système d'expérimentations de ROCKFlows se présente comme un exécutable. Cet exécutable permet l'exécution parallèle de *workflows* prédéfinis sur une même machine. Ce système d'expérimentations est basé sur un ensemble de jeux de données et un ensemble de *workflows* définis statiquement. Des métriques concernant l'exécution (ex. temps total) et la performance du *workflow* (ex. précision) sont récupérés et sauvegardés dans des tableurs. Ces derniers sont ensuite ajoutés dans la base de données des expérimentations à l'aide d'un script. Ces métriques sont ensuite utilisées pour prédire les performances de certains *workflows* et ainsi présenter à un utilisateur un portfolio trié en fonction de son problème.
La figure 3, ci-dessous, présente l'architecture globale de cette version 1.

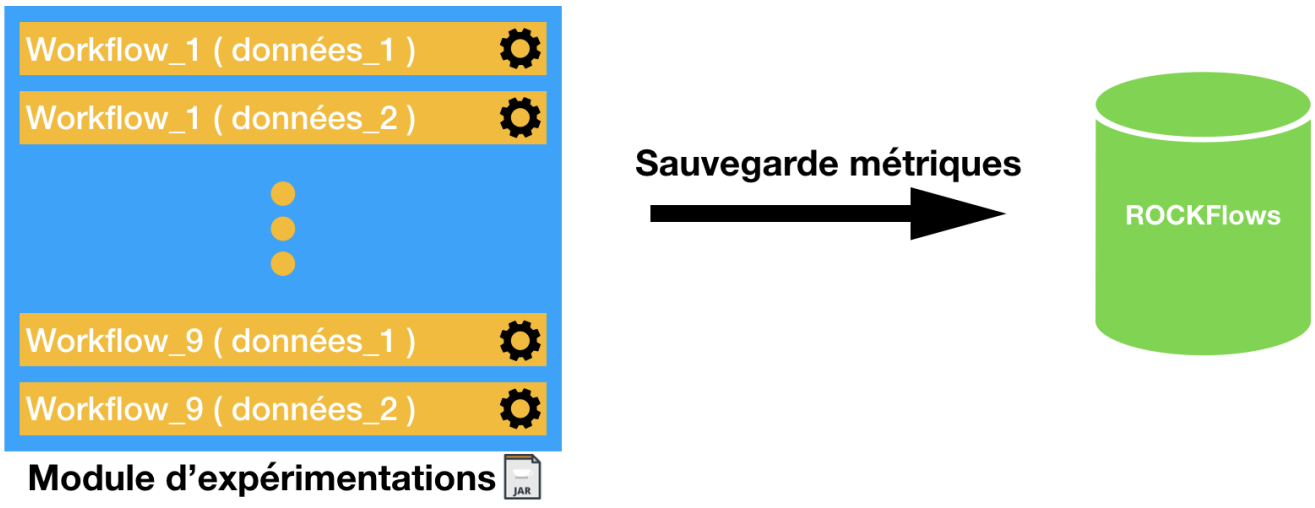


Figure 3: Architecture globale de la version 1 du système d'expérimentations.
A gauche l'exécutable contenant les *workflows* et les jeux de données. A droite la base de données des expérimentations. La flèche correspond au script de remplissage de la base de données.

Ce système présente des limites à différents niveaux. Cette partie liste ces différentes limites.

- **support multi-langage**. Le système ne supporte qu'une exécution mono-langage/mono-framework. Ceci entraîne une limite au niveau de la connaissance de ROCKFlows. En effet, il est important de pouvoir supporter un grand nombre d'algorithmes sans se soucier de leurs implémentations techniques (langage et dépendances) pour obtenir un portfolio diversifié correspondant aux *workflows* utilisés par la communauté de l'apprentissage automatique. Les algorithmes conçus par cette communauté étant développés sous des langages différents, il est nécessaire de supporter différentes plate-formes d'exécution.

- **automatisation de la récolte des métriques**. Le système sauvegarde les différentes métriques sous la forme de feuille Excels. Il existe un script pour remplir la base de données depuis ces feuilles excels, cependant ce procédé n'est pas automatique. Pour rester dans l'optique d'automatisation de la plate-forme il faut nécessairement que ce procédé soit automatique.

- **incrémentalité des exécutions**. L'ancien actuel ne permet pas de sélectionner un *workflow* à exécuter entraînant ainsi l'exécution de tous les *workflows* lors de l'ajout d'un élément. Ceci n'est plus envisageable étant donnée l'ordre de grandeur du nombre de *workflows*. Le nombre de *workflows* se compte en millions[7] et l'exécution totale prend donc un temps conséquent. De plus, en cas de problème système, il faudrait également tout relancer.

- **distribution des expérimentations**. L'exécution de l'ensemble des *workflows* se produit sur une unique machine. Pour la même raison que précédemment ceci n'est plus envisageable car le temps total d'exécution serait trop grand (des années de calculs). Il est donc nécessaire de pouvoir distribuer cette exécution.

- **composition automatique des expérimentations à mener**. Une des grandes problématiques de ROCKFlows est la composition de prétraitements[8] pour former des *workflows*. L'ancien système n'est pas branché à un moteur de composition et se base sur un ensemble restreint de processeurs. Ceci doit évoluer pour permettre l'exécution de l'ensemble des *workflows* possible pour un jeu de données.

- **gestion des erreurs**. L'ancien système ne gère pas les erreurs. Ce dernier doit au moins gérer les erreurs systèmes tels que des pannes.

- **contrôle des expérimentations**. Étant donné le temps total d'exécution de tous les *workflows* valides pour les jeux de données de référence de ROCKFlows, le fait de ne pas pouvoir facilement suivre cette exécution est contraignant.

[7] Le tableau 1 de la partie 3.1 présente le nombre de workflows par jeu de données (15 000 en moyenne). Pour 100 jeux de données cela donne 1 500 000 *workflows*.

[8] Dans le cadre de ROCKFlows, un processeur est soit un algorithme de pré-traitement, soit un algorithme d'apprentissage automatique

- **amélioration du relevé des métriques**. Finalement, nous nous sommes aperçu que nous n'exécutions qu'une seule fois un processeur donné. Nous avons besoin des plusieurs exécutions pour un algorithme donné enfin de récupérer des métriques de bonne qualité. Le système actuel ne le permet pas, le futur module doit donc améliorer le relevé des métriques.

2.3 Intégration du système de composition et d'expérimentations

Étant donné que des parties de ROCKFlows vont être changées, il sera alors nécessaire d'intégrer ces nouveaux systèmes à l'existant. Cette intégration doit être **iso-fonctionnelle**, c'est à dire que l'on doit être capable d'au moins proposer les mêmes fonctionnalités qu'auparavant.
Pour des raisons de passage à l'échelle et étant donné la nécessité de construire le portfolio sur un grand nombre (millions) d'expérimentations et les temps très longs (des mois en temps total) des expérimentations, il a été envisagé d'**intégrer des expérimentations issus d'autres systèmes**. Aussi, à travers cette intégration cherchons-nous également à aspirer de façon automatique des jeux de données et exécutions d'OpenML*100* [9] (lorsque ces exécutions sont comparables à nos exécutions), des algorithmes de Weka[10] (pré-traitements et algorithmes d'apprentissage automatique), et de nos anciennes exécutions.
A travers cette intégration, un travail de **simplification du déploiement** est aussi recherché. En effet, l'un des objectifs de cette intégration est d'obtenir un moyen rapide de déployer ROCKFlows.

2.4 ROCKFlows en tant que service

A travers mon apprentissage, nous avons pris connaissance d'un nouveau type d'utilisateur: Un utilisateur développeur qui souhaite utiliser de l'apprentissage automatique. Cet utilisateur n'est pas un expert en apprentissage automatique et utilise ainsi ROCKFlows pour obtenir un *workflow* à utiliser en fonction de son problème. La différence avec notre premier type d'utilisateur est l'interface utilisé. Le premier utilise une interface web alors que le second utilise un programme. La découverte de ce nouveau type d'utilisateur met en avant le besoin d'avoir ROCKFlows en tant que service. Il s'agit alors de concevoir un programme qui prend un jeu de données d'entraînement et un jeu de données à prédire et qui renvoie le jeu de données prédit et le *workflow* qui a été utilisé pour prédire ce jeu de données.

[9] OpenML100 contient des expérimentations sur 100 jeux de données de références : https://docs.openml.org/benchmark/#openml100
[10] Weka est une suites d'algorithmes d'apprentissage automatique : https://www.cs.waikato.ac.nz/~ml/weka/

3. Mes réalisations logicielles

Cette partie décrit le travail effectué lors de mon apprentissage après l'analyse de ROCKFlows et est décomposée suivant les objectifs décrit précédemment. La suite de ce rapport décrit ainsi mes réalisations logicielles après ma contribution sur les objectifs de ROCKFlows.
Chaque composant développé pendant mon apprentissage est exécutable sous forme de conteneur Docker[11]. La plupart des composants développés sont exécutables de façon indépendante et regroupés en système grâce à la technologie Docker-compose[12]. Cette précision est importante pour la bonne compréhension du déploiement et de l'intégration des différents composants présentés dans cette partie. L'utilisation de cette technologie répond au besoin de simplification du déploiement de la plate-forme.
L'annexe A contient un exemple de définition d'image Docker.

3.1 Composition de *workflows*

La composition de *workflows* est, comme énoncé précédemment, une grande problématique de ROCKFlows. Pour pouvoir exécuter des expérimentations dans un temps raisonnable (quelques mois), il est nécessaire de réduire l'espace des *workflows*. Pour ce faire, il est tout d'abord nécessaire de réduire cet espace à l'espace des *workflows* qu'il est possible de réaliser. Il s'agit ensuite de définir des contraintes métier permettant de réduire cet espace de manière cohérente et justifié. Cette partie présente les informations que nous pouvons extraire des algorithmes d'apprentissage automatique ainsi que la solution que j'ai mis au point lors de mon apprentissage

Chaque algorithme possède des préconditions d'exécution. On appelle conditions, les limites d'exécution défini par un algorithme. Par exemple l'algorithme `Normalisation` a besoin de données numérique. Pour permettre l'automatisation de la vérification des préconditions, des méta-informations sont extraites d'un jeu de données. Ces méta-informations, comme par exemple `NombreInstancesNumerique`, correspondent avec les différentes conditions. Ces conditions se transcrivent sous la forme : `Normalisation(NombreInstancesNumerique > 0)`.
De plus, pour permettre de capturer la sémantique des pré-traitements, des postconditions ont été introduites. Par exemple, le prétraitement `TransformationNumériqueEnNominal` possède comme postconditions `NombreInstancesNumerique = 0 et NombreInstancesNominal > 0`.
Ces deux types de conditions permettent ainsi de composer de façon automatique les *workflows* valide syntaxiquement. Cependant, ces conditions ne sont pas suffisantes pour restreindre l'espace des *workflows* (cf. présentation des résultats de la restriction de l'espace). Des contraintes entre algorithmes doivent être mises en place. Ces contraintes inter-algorithmes permettent de capturer des informations sur la sémantique des agencements des *workflows*. Ces dernières sont ainsi différentes des conditions définies de manière absolue par un algorithme. Par exemple, il n'est pas cohérent d'enlever les valeurs manquantes après avoir exécuté des pré-traitements. Ces contraintes

[11]Docker est une technologie de contenerisation d'application : https://docs.docker.com/engine/reference/run/
[12] Docker-compose est un outil permettant d'intégrer différents conteneurs Docker: https://docs.docker.com/compose/

inter-algorithmes sont définies par un expert et permettent de définir un ordre partiel entre les pré-traitements.

L'espace des *workflows* que nous cherchons à générer pour un problème donné peut ainsi être créé grâce à des pré/post-conditions et des contraintes inter-algorithmes. Durant mon apprentissage j'ai ainsi développé un moteur de composition qui se base sur ces critères-là. Ce moteur fonctionne à l'aide d'un graphe. Chaque noeud correspond à un algorithme et deux noeuds sont reliés si ces derniers sont valides (syntaxiquement et sémantiquement) deux à deux (pour tous les jeux de données). Il s'agit donc de trouver les chemins/*workflows* valides syntaxiquement et sémantiquement. Ce modèle en graphe nous permet de ne pas visiter tous les workflows lors de la génération. En effet si un bout de chemin est invalide, il est alors immédiatement oublié. Ceci permet de limiter la complexité (complexité linéaire au nombre de *workflows* valides) et ainsi de mieux passer à l'échelle. Ce modèle est incrémental, et permet de passer à l'échelle en termes d'ajout de nouveaux algorithmes en limitant la vérification des contraintes aux contraintes entre deux algorithmes.
Le tableau 1 présente le résultat de cette réduction de l'espace et le tableau 2 présente la performance de ce système de composition (en termes de complexité).
Ces deux tableaux sont basés sur un ensemble de 483 algorithmes d'apprentissage automatique et 9 pré-traitements. Le nombre total de *workflows* (sans appliquer de conditions/contraintes) est de 670 758 800 (nombre total d'agencements possible).

Tableau 1: Nombre de workflows en fonction des conditions/contraintes et jeux de données.

Jeu de données	**En utilisant seulement les pré/post conditions**	**En utilisant les conditions et les contraintes**
Aucun	3 653 644	39 694 *(nombre maximum de workflows valide)*
Speed-dating[13]	3 419 170	36 360
Texture[14]	256 286	5 688
Iris[15]	105 892	3190

Tableau 2: Nombre de workflows visités et valide en fonction des jeux de données.

Jeu de données	**Nombre de workflows visités**	**Nombre de workflows valide**
Aucun	61 720	39 694
Speed-dating	56 788	36 360
Texture	14 088	5 688

[13] https://www.openml.org/d/40536
[14] https://www.openml.org/d/40499
[15] https://www.openml.org/d/61

Iris	8 320	3190

Comme nous pouvons l'observer, le nombre de *workflows* visités par l'algorithme de composition est assez proche (facteur 2) du nombre de workflows valides. Cela signifie que nous visitons principalement que des workflows validant les conditions et contraintes inter-algorithme. La complexité calculée nous donne environ un facteur de 2 pour l'ensemble de nos jeux de données, amenant ainsi la complexité de la composition au nombre de *workflows* cherché, contre une complexité factorielle avec une approche triviale. En effet l'approche triviale consisterait à visiter tout les *workflows* et de vérifier leur validité un par un. Le nombre total de workflows est factoriel par rapport au nombre d'algorithmes. Ce nombre correspond au nombre d'agencements possible de *workflows*. Ceci nous a permis de valider la performance de ce modèle.

Une approche utilisant un solveur de contraintes a été testé mais ne passait pas à l'échelle pour l'ensemble des prétraitements. Je ne suis pas arrivé à modéliser ce problème de manière efficace en utilisant des outils dédiés à la résolution de contraintes. Ce problème n'est pas trivial et demande de grandes compétences de modélisation.

Ce modèle en graphe possède également un autre atout. Il est évolutif dans l'ajout de nouveaux algorithmes et de stratégie de visite du graphe. Nous verrons par la suite que ce modèle sera amené à être utilisé par un autre composant de ROCKFlows.

3.2 Système d'expérimentations

Le nouveau système d'expérimentations doit répondre à ces différents besoins : Il doit permettre une exécution distribuable, générique et contrôlé. Le système doit être capable de repérer et de traiter les erreurs d'exécution. Le système doit être résilient en cas de panne.
Une partie de ce système d'expérimentations a été développé lors de la première partie de mon apprentissage, cependant des améliorations et des changements ont été apportés. En l'occurrence, nous avons décidé de séparer ce système en deux. Ces systèmes seront ainsi présentés par ordre de dépendance.
Ce système d'expérimentation doit permettre d'évaluer un ensemble de *workflows* sur un ensemble de jeux de données pour ainsi obtenir des métriques telles que la précision ou la surface sous la courbe ROC[16]. Ces métriques correspondent aux performances des *workflows*. La précision et la surface sous la courbe ROC sont deux indices de performances très utilisés pour comparer des *workflows*. La méthode d'évaluation utilisée ici est le *10-folds*[17]. Cette méthode divise un jeu de données en deux parties : 90% pour de l'entraînement et 10% pour du test de 10 façons différentes. Ceci entraîne donc la création de 10 plis (i.e. *folds*) contenant un jeu de données pour l'entraînement et un jeu de données pour le test.

3.2.1 Planificateur de tâches

A travers différentes expérimentations, nous avons décidé d'utiliser la technologie Docker pour encapsuler les exécutions de nos différentes expérimentations (cf. Annexe A. Encapsulation avec Docker). Nous définissons ainsi chaque algorithme par une image Docker et des paramètres d'exécution. Le listing 1 présente un exemple de définition d'un algorithme :

```
{
    "processType": {
      "type": "algorithm"
    },
    "name": "weka-J48",
    "image": {
      "name": "rockflows/algorithm-weka"
    },
    "options": [
      "-e CLASSIFIER=weka.classifiers.trees.J48"
    ]
}
```

Listing 1: Exemple d'algorithme.

L'algorithme de la figure 4 doit être exécuté avec l'image `rockflows/algorithm-weka` (image Docker correspondant à une exécution générique d'un algorithme Weka) avec les options `-e`

[16] https://fr.wikipedia.org/wiki/Courbe_ROC
[17] https://www.openml.org/a/estimation-procedures/1

`CLASSIFIER=weka.classifiers.trees.J48` (options permettant de définir quel algorithme de weka à utiliser).

Ce planificateur de tâches permet ainsi d'exécuter des tâches Docker. On peut par exemple exécuter un *workflow* pour un certain jeu de données.
La logique d'exécution de ce planificateur permet de gérer les dépendances entre tâches : les tâches issues d'un *workflow* sont exécutées dans le bon ordre.
Ce planificateur est aussi résilient à la panne. En effet, chaque modification de l'état du planificateur est sauvegardée en base de données. Le planificateur peut donc être reconstitué lors d'un arrêt ou d'une panne.
A la fin de chaque exécution d'une tâche, des métriques système (temps cpu, ram utilisée) et métier (précision d'un algorithme d'apprentissage automatique) sont insérés dans un registre de métriques. Ce registre est couplé à un aggrégateur de métriques et permet ainsi d'obtenir les résultats des expérimentations lancées à travers ce planificateur.
De la même façon que les métriques sont sauvegardées, les journaux standard et d'erreurs ainsi que les codes de retours des exécutions sont également sauvegardés ce qui permet de repérer de potentielles erreurs, tel que l'oublie du précondition d'un algorithme, de façon automatique. Si une erreur est déclenchée, les tâches qui dépendent de la tâche en erreur ne seront pas lancées. Nous avons également décidé d'intégrer un système de monitoring appelé Prometheus[18] sur le planificateur de tâches. Ceci permet de s'assurer du bon fonctionnement du planificateur à chaque instant. Cela permet également d'avoir un suivi graphique des exécutions en utilisant des outils tels que Grafana[19].
Une autre caractéristique de ce planificateur est l'idempotence des tâches. Cette caractéristique permet de limiter certaines exécutions. Les tâches du planificateur sont idempotentes. L'exécution de deux fois la même tâche est équivalent à l'unique exécution de cette même tâche. Ceci permet d'exécuter qu'une seule fois une même tâche et permet de diminuer la complexité d'exécution des *workflows*. En effet, si le *workflows* `A;B;C` pour le jeu de données 'd' a déjà été exécuté, alors l'exécution du *workflow* `A;B;D` entraînera l'unique exécution de D pour le jeu de données 'd' modifié par A puis B. Pour se faire les résultats intermédiaires sont sauvegardés.
Les exécutions des différentes tâches se font à travers des exécuteurs. Ces exécuteurs peuvent avoir des ressources limites. Dans le cadre de nos expérimentations, nous limitons ces ressources de façon uniforme pour obtenir des résultats comparables. Les différentes expérimentations sont toutes lancés en utilisant la même taille mémoire et puissance de calcul. Ces exécuteurs peuvent être placés sur des machines différentes à condition d'y spécifier un gestionnaire de volume distribué. Ceci permet de rendre ce planificateur distribuable pour passer à l'échelle.
La figure 4 présente l'architecture de ce planificateur de tâches.

[18] Prometheus est un système de monitoring : https://prometheus.io/
[19] Grafana est un outil de visualisation de séries temporelles. Ce dernier peut intégrer les résultats capturés par Prometheus : https://grafana.com/

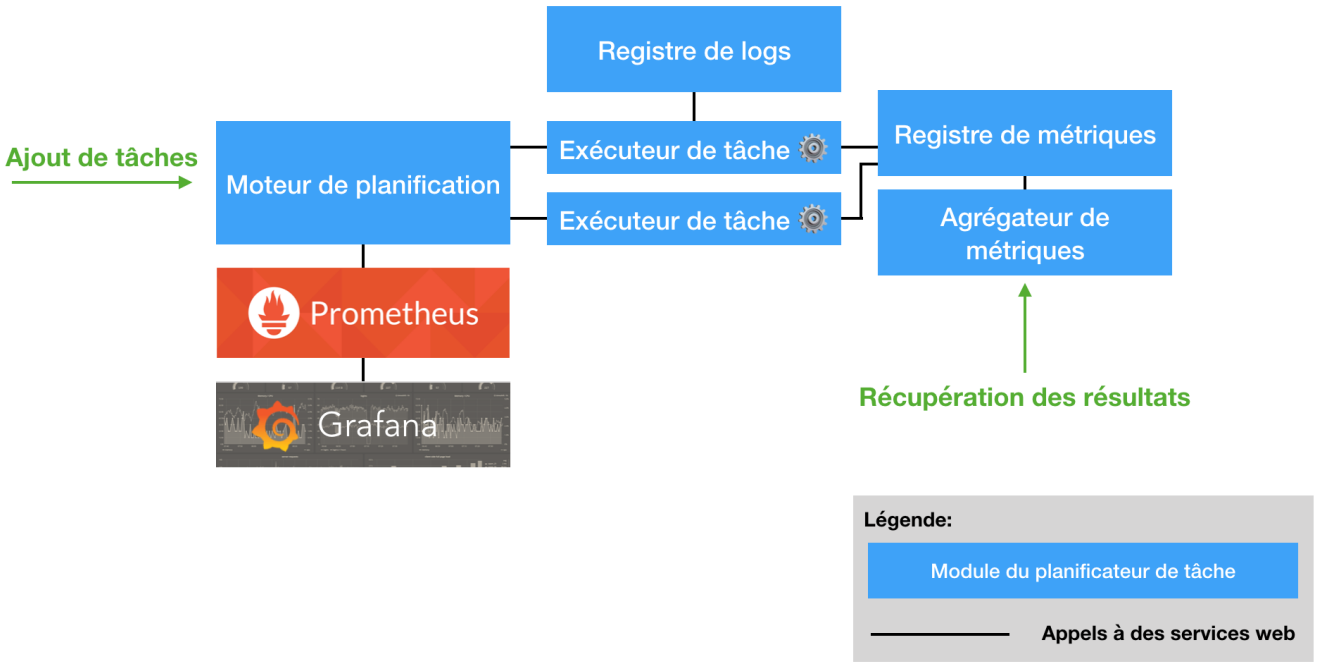


Figure 4: Planificateur de tâches.

Les tâches sont ajoutées au moteur de planification et les résultats sont récupérés à travers l'agrégateur de métriques.

3.2.2 Système d'expérimentations

Le système d'expérimentation utilise le planificateur de tâches (en version distribuée) pour l'exécution des expérimentations. Ce système est basé sur des registres et des évènements. La figure 5 représente le flot principal de ce système.

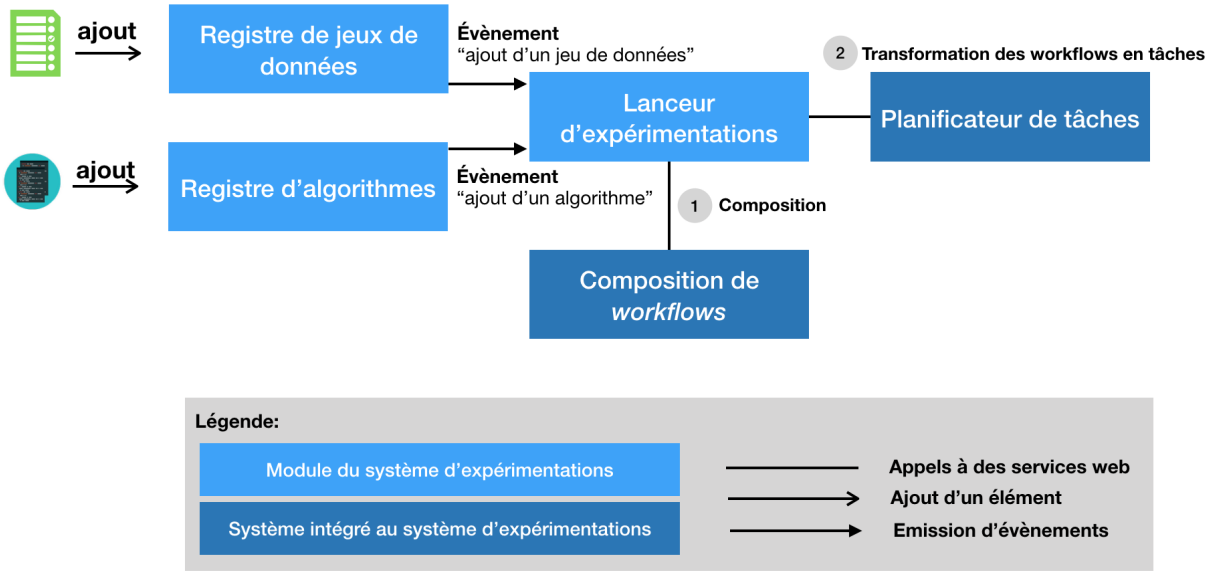


Figure 5: Système d'expérimentations.

Ce système possède deux types de registre, un pour chaque type d'élément d'expérimentation. Le registre de jeux de données permet l'enregistrement et le stockage des différents jeux de données dédiés aux expérimentations et le registre des algorithmes permet quant-à-lui l'enregistrement des algorithmes avec leur méthode d'exécution (cf. figure 4) et leurs conditions/contraintes. Lors de l'ajout d'un élément à l'un de ces deux registres, un événement est déclenché.
Le composant "`Lanceur d'expérimentation`" écoute ces différents évènements et génère les *workflows* à exécuter grâce au système de composition de *workflows*. Ce composant permet ainsi de transformer ces *workflows* en tâches grâce au "`Convertisseur en tâche`". Ce composant convertit les *workflows* en tâches en ciblant la technique d'évaluation *10-folds*. Ces tâches sont ainsi envoyées au planificateur de tâches qui possède une file permettant de temporiser l'exécution des tâches. Ceci permet, entre autres, de ne pas surcharger le planificateur de tâches lors d'un ajout conséquent d'éléments au système. Les résultats des expérimentations sont ensuite stockés dans le registre de métriques du planificateur de tâches.
Cette architecture basée sur une communication par événements permet d'intégrer de façon souple différents systèmes et répond ainsi aux problématiques de passage à l'échelle et d'évolutivité. En effet, il est assez simple de changer le comportement des expérimentations en venant s'intercaler sur les événements ou sur le lanceur d'expérimentations.
Le planificateur de tâches utilisé au sein du système d'expérimentation est une version modifiée de celui présenté précédemment. ROCKFlows utilise les résultats des expérimentations en tant que base d'apprentissage pour prédire la performance des *workflows* valides pour un problème donné. L'une des idées ici est ainsi d'exécuter les workflows dans un ordre qui permet d'augmenter la qualité de l'apprentissage le plus vite possible (i.e. on cherche à apprendre le plus vite possible). Pour ce faire, nous ajoutons une logique supplémentaire au planificateur de tâches qui est de choisir quelle tâche exécuter à un instant donné pour répondre à ce problème. La version actuelle ne répond pas encore à cette problématique et choisit le premier *workflow* par ordre d'arrivée mais cette modification permet d'anticiper cette fonctionnalité. C'est ici que le modèle en graphe de la composition de *workflows* sera amené à être utilisé en utilisant un nouvel algorithme reposant sur ce modèle de graphe. A la place de parcourir les *workflows* valides, cette nouvelle stratégie de recherche permettra de trouver le prochain *workflow* à exécuter. Cette stratégie de recherche remplacera ainsi la stratégie de visite pour cette répondre à cette problématique.

3.2.3 Une interface graphique pour le suivi des expérimentations

Pour permettre à un utilisateur expert de suivre l'exécution des expérimentations, un interface minimaliste a été réalisée. Cette interface couvre les fonctionnalités principales du système d'expérimentations. Un utilisateur peut ainsi suivre en direct l'exécution des *workflows*, obtenir des informations sur les registres tels que la liste des jeux de données avec un moyen de les télécharger, la liste des différents algorithmes et les résultats des expérimentations.
En annexe B se trouvent différentes captures d'écran présentant cette interface.

3.2.4 Un *Langage Spécifique au Domaine* (LSD) pour les expérimentations

Après avoir mis au point ce système d'expérimentation, nous avons mis en évidence le besoin d'un utilisateur expert à pouvoir tester un *workflow* particulier pour un certain jeu de données. Pour

répondre à cette problématique j'ai proposé de définir un *Langage Spécifique au Domaine* (LSD). Ce langage permet ainsi d'exécuter un certain *workflow* pour un jeu de données et d'obtenir les résultats de cette exécution. Ce LSD utilise directement le système d'expérimentation et possède ainsi les propriétés de ce dernier tels que l'idempotence.
Le listing 2 présente un exemple de script utilisant ce LSD.

```
# On envoie le jeu de données au registre s'il n'est pas encore défini
my_dataset = upload './dataset' as 'mon_jeu_de_donnees'

# On déclare notre workflow (on exécute le pre-traitement 'remove-missing-values'
suivi de weka-J48 avec le jeu de données défini plus haut)
my_workflow = my_dataset -> remove-missing-values;weka-J48

# On évalue notre workflow en utilisant le méthode 10-folds
evaluate my_workflow using 10-folds as 'Mes-resultats'

# On capture les logs de l'exécution du workflow
my_logs = logs of my_workflow

# On capture les métriques de l'exécution du workflow
my_metrics = metrics of my_workflow

# On affiche les logs et les métriques
print my_logs
print my_metrics

# On sauvegarde les métriques dans un fichier
save my_metrics to './metrics.json'
```

Listing 2: Un exemple du LSD d'expérimentations pour l'évaluation d'un workflow.

Ici, la méthode d'évaluation 10-folds est celle utilisée au sein du système d'expérimentation de ROCKFlows. Ce LSD supporte différentes techniques d'évaluation et peut ainsi, potentiellement, être utilisé pour lancer des expérimentations dans des domaines différents de l'apprentissage automatique. Nous sommes sur ce point en collaboration avec une équipe dirigé par Claude Pasquier[20] travaillant sur des expérimentations sur des brins d'ADN, et l'un des objectifs est d'utiliser ce moteur d'expérimentation pour lancer leurs expérimentations.

Il est possible de lancer le système d'expérimentations en mode bac-à-sable. Ceci permet ainsi de tester l'exécution de certains *workflows* afin de vérifier leur bon fonctionnement (fonctionnement sans erreur). Si une erreur est détectée, l'exécution de ce script renverra une erreur.
L'exécution de scripts du LSD hors mode bac-à-sable entraîne une exécution à haute priorité. Ceci permet ainsi d'exécuter un certain *workflow* sans avoir à attendre l'exécution des *workflows* déjà présents dans la file d'attente du planificateur de tâches.

Ce LSD permet également de lancer l'exécution d'un *workflow* sans évaluation et d'en sauvegarder le résultat dans un dossier. Ceci permet, par exemple, d'entraîner un *workflow* et de prédire grâce à ce *workflow* entraîné les classes d'un jeu de données cible (dans le cadre d'un workflow destiné à de la classification). Le listing 3 présente un exemple de cette utilisation du LSD.

[20] http://i3s.unice.fr/~cpasquie/

```
# On définit notre jeu de donnée (le dossier contient train.arff et
toPredict.arff)
my_dataset = upload './dataset_to_predict' as 'my-dataset'

# On défini notre workflow. Ici on donne des paramètre à l'algorithme weka-J48
my_workflow = my_dataset -> weka-J48(-S, -L, -Q=123)

# On exécute le workflow et on capture les résultats
my_results = run my_workflow

# On affiche les logs et les métriques système de cette exécution
print logs of my_workflow
print metrics of my_workflow

# On sauvegarde les résultats de l'exécution
save my_results to './dataset_out'
```

Listing 3: Un exemple du LSD pour l'exécution d'un workflow.

3.3 Intégration du système de composition et d'expérimentations

Pour intégrer ces nouveaux systèmes à l'interface web de ROCKFlows, et ainsi permettre aux utilisateurs d'obtenir de meilleures prédictions, il est nécessaire de définir les points où ces nouveaux systèmes doivent être intégrés.
La figure 6 présente ces différents points. En plus de l'intégration de ces nouveaux systèmes, des résultats d'expérimentations produits en dehors du nouveau système[21] d'expérimentations (tels que les anciennes expérimentations et les expérimentations de OpenML*100*) doivent être intégrés. Notons ici que nous intégrons uniquement ce qui est comparable[22] aux expérimentations que nous pouvons réaliser,

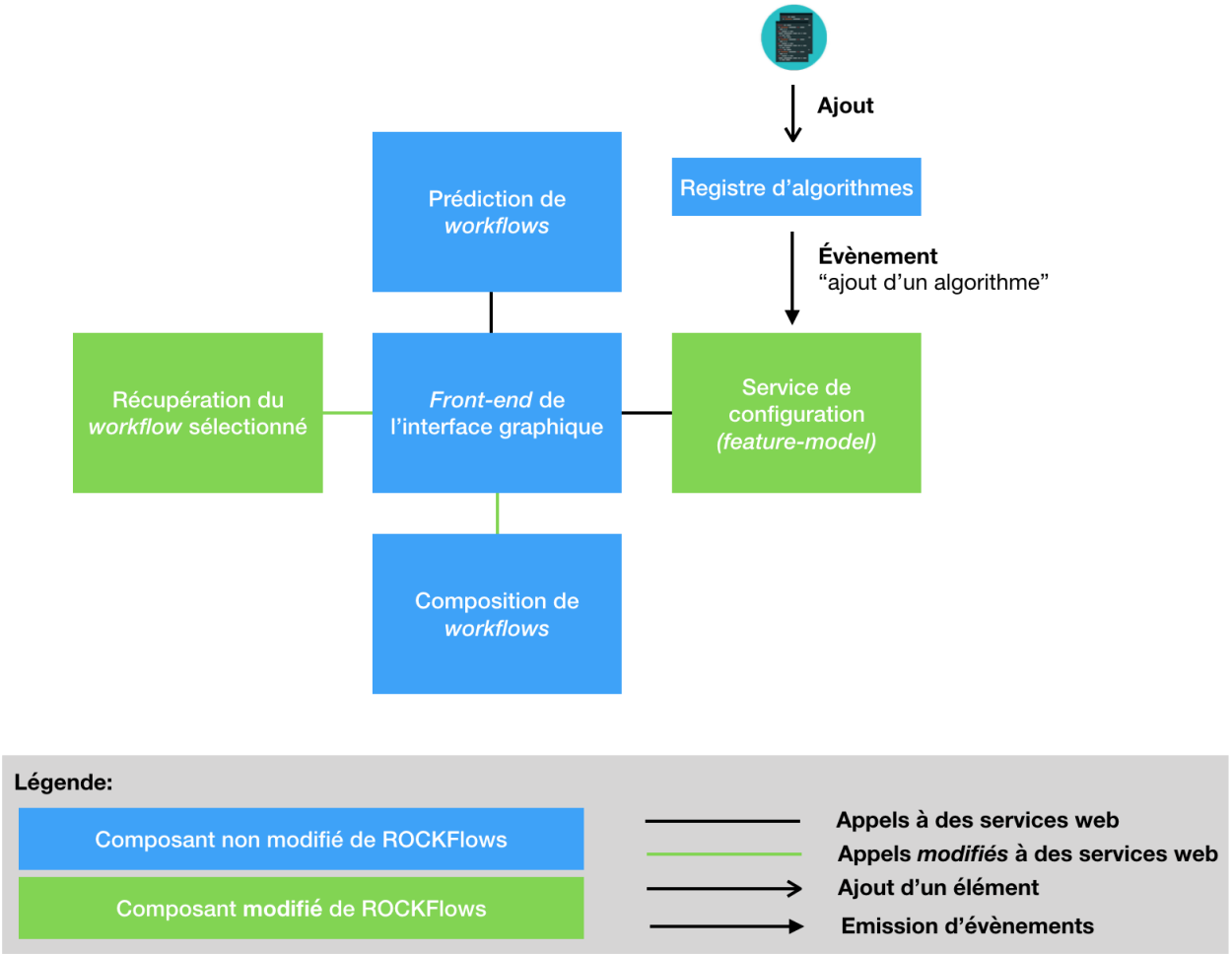


Figure 6: Points d'intégration de l'interface web.

[21] Nous appellerons ces expérimentations, des *expérimentations externes*

[22] Nous considérons comme comparables des expérimentations de classification réalisées en utilisant une évaluation 10 folds, pour lesquelles nous avons les codes et pour lesquelles nous avons "confiance" dans les résultats, ce point est abordé au 3.3.2.

3.3.1 Intégration des nouveaux systèmes

Pour permettre la sélection d'un *workflow* parmi le portfolio, l'utilisateur décrit son problème à travers une interface graphique. Cette interface, après avoir extrait les méta-informations du jeu de données de l'utilisateur, fait appel au **moteur de composition** pour générer les *workflows* correspondant à son problème. Par la suite, le **module de prédiction** est en charge de prédire les performances[23] de ces différents *workflows*. Un module de configuration est ensuite utilisé pour permettre à l'utilisateur de choisir un *workflow*. Ce *workflow* est ainsi fourni en tant qu'**exécutable modifiable** à l'utilisateur. Ce module de configuration permet ainsi de sélectionner un *workflow* en fonction des choix de l'utilisateur comme une haute précision avec un temps d'exécution haut tout. Cette configuration permet de fournir à l'utilisateur un choix de sélection en fonction de diverses contraintes.
Il s'agit donc ici de remplacer ou de modifier certains de ces modules.

- **moteur de composition**. Pour intégrer le système de composition développé lors de mon apprentissage, il a suffi de modifier l'interface graphique pour que cette dernière effectue les requêtes correspondant au nouveau système.

- **module de prédiction**. Ce module a dû être modifié au niveau de l'entraînement. En effet, ce module doit utiliser la nouvelle base d'expérimentations. Ce module fait donc maintenant appel au registre des jeux de données, et pour chacun d'entre eux, récupère les *workflows* valides grâce au moteur de composition. C'est sur les résultats enregistrés dans le registre de métriques que le système s'entraîne à prédire. En plus de cette modification de la base d'entraînement, une légère modification du modèle d'apprentissage a été réalisée. En effet, pour obtenir de meilleurs résultats, nous avons décidé de mieux représenter les différents pré-traitements de chaque *workflow*. Précédemment, nous gérions les pré-traitements comme des identifiants regroupés. A travers cette intégration, ces identifiants ont été séparés pour chaque pré-traitement pour permettre à l'algorithme de prédiction de mieux fonctionner.

- **module de configuration**. Pour permettre l'évolution incrémentale de la plate-forme, il est nécessaire de modifier le module de configuration à chaque ajout d'algorithme. Pour ce faire, un intégrateur a été mis en place. Ce dernier écoute les événements déclenchés lors de l'ajout d'un algorithme au registre d'algorithmes. Dès qu'un algorithme est ajouté, le module de configuration est changé pour inclure celui-ci. Cette partie d'intégration a été plus compliquée à réaliser car il a été nécessaire de bien comprendre le fonctionnement de ce dernier pour effectuer ces changements. Une phase de rétro-ingénierie a donc été nécessaire avant cette intégration.

- **exécutable modifiable**. Le résultat attendu par un utilisateur de l'interface web de ROCKFlows est un *workflow* exécutable et modifiable. Ce workflow se présentait, avant mon apprentissage,

[23] La précision, le temps moyen d'exécution, l'espace mémoire utilisé et la surface sous la courbe ROC sont des indices de performances des *workflows*

comme un projet Java. Cependant, suite à l'évolution de la plate-forme, et notamment suite à la prise en compte d'algorithmes multi-langages, cette partie a dû être modifiée. Nous fournissons dorénavant un dossier contenant le workflow à exécuter. Ce dossier contient le workflow sous la forme d'un script du LSD présenté en 3.2.4. L'exécution de ce *workflow* se fait donc à travers le planificateur de tâches lancé sur la machine de l'utilisateur.

3.3.2 Intégration des *expérimentations externes*

Pour augmenter la qualité de la prédiction, il est nécessaire d'augmenter la base d'apprentissage. Pour rester iso-fonctionnel sur la qualité de la prédiction, il est important d'utiliser les résultats des expérimentations réalisées à travers l'ancien système d'expérimentations. C'est à l'aide d'un script de conversion que nous avons procédé. L'ancienne base d'expérimentation a ainsi été converti puis ajouté au registre de métriques de nos nouvelles expérimentations.
Un autre moyen d'augmenter notre base d'expérimentations est d'ajouter des expérimentations réalisées par des tiers parties. Nous nous sommes ainsi intéressés avec François Montigny à aspirer des expérimentations de OpenML*100*. Avant de faire ceci, il a été nécessaire de sélectionner les résultats qui correspondent à notre base d'algorithmes et à vérifier que les résultats fournis par OpenML*100* sont comparables aux nôtres. François Montigny, un stagiaire qui a travaillé sur ROCKFlows pendant cet été, a généré des scripts du LSD présentés en 3.2.4 sur un sous ensemble de résultats représentatifs. Une comparaison automatique a révélé que les résultats des *workflows* sous Weka de OpenML*100* sont comparables à nos expérimentations. Son travail nous a ainsi permis d'intégrer les résultats des *workflows* utilisant Weka d'OpenML*100*.

3.4 ROCKFlows en tant que service

A travers mon apprentissage, nous nous sommes aperçu de l'existence d'un nouveau type d'utilisateur : l'utilisateur développeur qui souhaite utiliser de l'apprentissage automatique. On retrouve ce type d'utilisateur dans les approches d'auto-apprentissage automatique tel que *auto-sklearn*[24]. Dans cette approche l'utilisateur se sert d'une interface masquant le travail de recherche ou d'optimisation du workflow répondant au problème de l'utilisateur.

Dans le cadre de ROCKFlows, l'idée est de proposer à ce type d'utilisateur un programme permettant de répondre à son problème de façon transparente. Dans le cadre de la classification, l'utilisateur fournit en entrée son jeu de données d'entraînement et celui à prédire et il obtient en sortie le jeu de données prédit ainsi que le *workflow* qui a été utilisé. Par exemple, l'utilisateur fournit une base d'images avec des labels chien ou chat (entraînement) et des images non labellisées (à prédire). Le système retour ainsi les images avec les labels prédits ainsi que le *workflow* qui a été utilisé pour labéliser les images de chat/chien. Mon travail sur cette partie se résume donc à intégrer les différents systèmes de ROCKFlows pour en obtenir une version en ligne de commande. Ceci s'est fait en développant un intégrateur faisant appel aux différents composants de ROCKFlows de manière similaire à l'interface web. L'une des différences majeures avec l'interface web est que ce processus est complètement automatique et ne laisse pas choisir à l'utilisateur le *workflow* en fonction des différentes performances. Le *workflow* est sélectionné selon les critères choisis par l'utilisateur (ex. *workflow* avec la meilleur précision).

La figure 6 présente l'architecture de ROCKFlows en tant que service. Le listing 4 présente une exécution type de ce service.

[24] https://github.com/automl/auto-sklearn

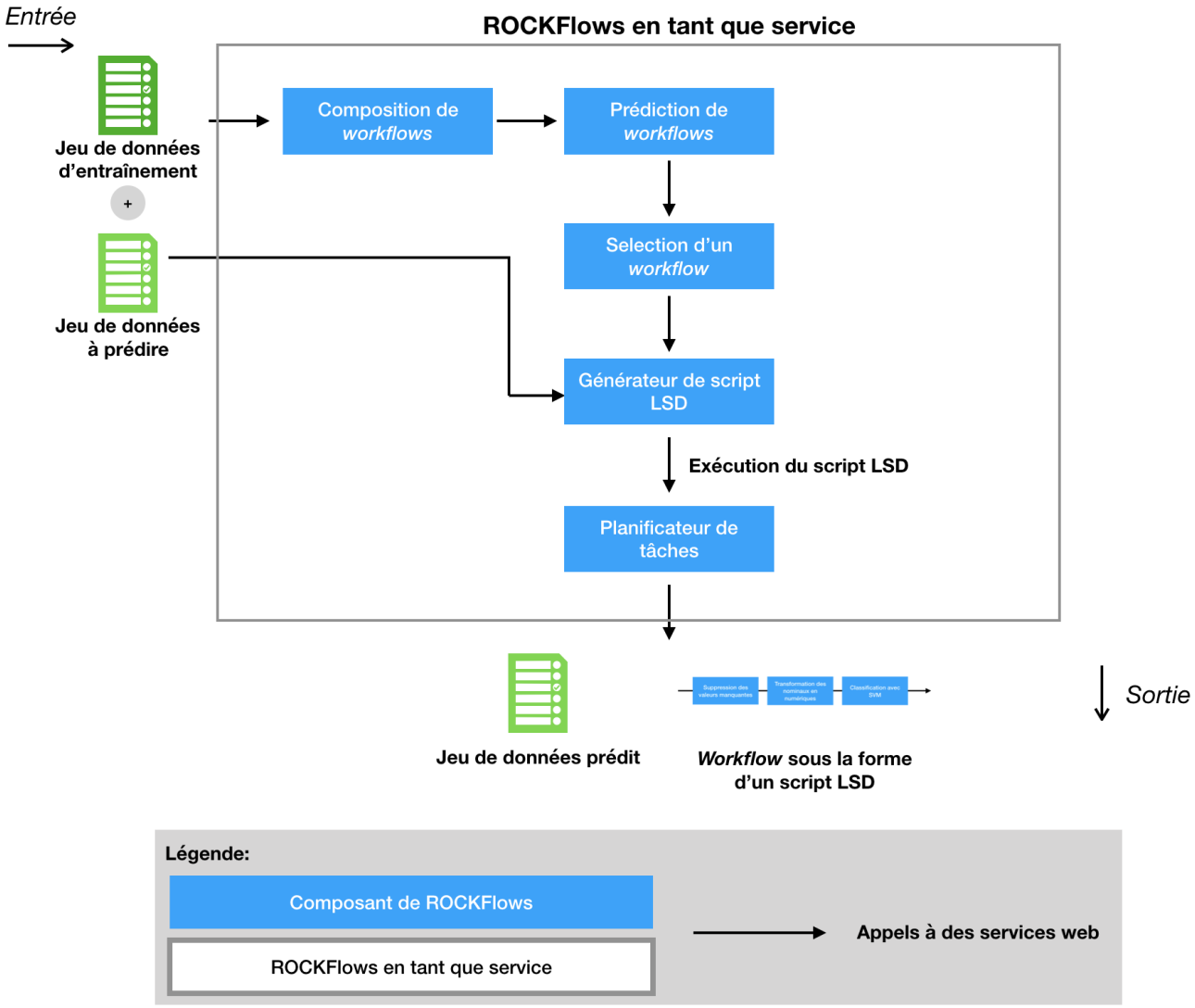


Figure 6: Architecture de ROCKFlows en tant que service.

```
$ ./rockflows my-dataset ./input/ ./output/
Extracted the meta-features of the dataset my-dataset
Found 2380 workflows
Predicted the accuracy of each workflow
Selected the workflow normalize;weka-RandomForest with predicted accuracy of
76.30366029089497
Executing workflow...
Workflow executed. Results saved to ./output/
```

Listing 4: Exécution sous ligne de commande de ROCKFlows.

Le listing 4 correspond à l'exécution de ROCKFlows en tant que service pour le jeu de donnée nommé `my-dataset` situé dans le dossier `./input`. Le dossier `./output` contient, en fin d'exécution, le jeu de données prédit ainsi que le *workflow* utilisé sous la forme d'un script du LSD présenté en 3.2.4.

4. Synthèse des résultats

Au cours de mon apprentissage, j'ai fait évoluer le projet ROCKFlows sur plusieurs axes. J'ai, à travers cet apprentissage, travaillé en fonction des objectifs donné par l'équipe ROCKFlows ainsi que les utilisateurs rencontrés de façon autonome. La plupart des modifications de la plate-forme sont issue de ma compréhension des objectifs à long et court terme de ROCKFlows. J'ai ainsi redéfinie l'architecture de ROCKFlows pour répondre aux objectifs d'automatisation, validation, incrémentalité et passage à l'échelle. Je ne reviens pas sur ces différents points pour mettre le focus sur quelques résultats particulièrement significatifs.

Automatisation du déploiement - Tout d'abord, en termes de déploiement, ROCKFlows est déployable en une ligne de commande grâce à la technologie Docker.
Les différentes parties indépendantes de ROCKFlows tels que le planificateur de tâche sont elles aussi déployables en une ligne de commande. Ceci permet ainsi de réutiliser certains des systèmes développés pendant mon apprentissage de façon indépendante. Une des utilisations externes à ROCKFlows est l'utilisation du planificateur de tâches Docker avec le LSD par une équipe de recherche travaillant sur des expérimentations sur des brins d'ADN.

Construction incrémentale de l'espace des workflows - Pour répondre à la fois aux objectifs de validation et de passage à l'échelle, les travaux menés sur le système d'expérimentation ont permis d'une part d'automatiser la construction des workflows en fonction des nouveaux algorithmes et d'autre part de réduire cet espace aux seuls workflows valides et présentant de la valeur. Cette réduction permet d'obtenir un portfolio de *workflows* de taille réaliste et justifiable. Ce faisant, nous réduisons les temps d'exécution aux expérimentations sensées être pertinentes.
A l'heure actuelle, des expérimentations sont toujours en cours d'exécution et leurs résultats seront intégrés à ROCKFlows dès que possible. Le système est bien iso-fonctionnel, ce qui correspond à un objectif de mon apprentissage.

Et finalement, l'aspect **ROCKFlows en tant que service**, que j'ai proposé et qui a provoqué l'enthousiasme de l'équipe permet d'une part de lancer et d'exécuter l'intégralité de ROCKFlows sur sa machine, mais d'autre part prépare le terrain pour une évolution possible de la plate-forme : utiliser ROCKFlows pour prédire quel type de *workflow* utiliser dans le module de prédiction de *workflows*, "bootstrappant" ainsi le système et laissant envisager des meilleures performances encore.

5. Perspectives

Les différents objectifs développés lors de cet apprentissage ont permis à ROCKFlows d'obtenir des propriétés de résilience, de suivi ainsi qu'une ouverture potentielle à de nombreuses perspectives.
Parmi celles qui sont particulièrement liées à mon travail, je noterais :
1. comparer "ROKCFlows as a Service" avec ScikitLearn[25]; quoique proposant une approche complètement différente, nous pourrions comparer la pertinence des résultats proposés par les 2 approches ;
2. mettre en place des stratégies d'expérimentation, basées sur une boucle de feedback sur les résultats obtenus ; l'architecture que j'ai mis en place devrait permettre l'introduction de tels composants;
3. générer des diagrammes de justification pour expliquer les processus ayant conduit aux prédictions ; ; la communication par évènement mise en place offre la modularité nécessaire à ce type d'extension.

6. Conclusion

En conclusion, mon apprentissage a permis de faire évoluer la plate-forme ROCKFlows tout en contribuant au développement de nouveaux objectifs. Cela m'a permis d'approfondir mes bases sur les architectures distribuées et leurs problématiques. Ce travail m'a également permis de mieux comprendre et développer des architectures basées sur des composants utilisants des interactions à base d'événements.
Le fait d'avoir travaillé sur un projet au sein d'un laboratoire de recherche m'a également permis de développer une vision scientifique du projet et et m'a permis d'appréhender des problèmes d'assez grande complexité. Ce point-là était très passionnant et m'a forcé à prendre du recul sur diverses situations. Je me rappelle encore du jour où, en discutant avec une partie de l'équipe, le temps de calcul estimé pour lancer les expérimentations était beaucoup trop grand car l'espace des *workflows* n'était pas assez réduit. Cette problématique était de taille et semblait difficile à aborder. Mais, à l'aide des différentes idées suggérées par l'équipe et de bonnes séances de réflection, ce problème a pu être pallié et a ainsi permis de transformer un temps de calcul estimé à des milliers d'années en quelques mois. Ceci est un des exemples qui m'a permis de réaliser l'importance de prendre du recul scientifiquement et de ne surtout pas abandonner face à un problème de haute taille.
Cet apprentissage m'a également permis de travailler dans un milieu de recherche et d'ainsi découvrir différents aspects de ce type de travail. Cette apprentissage s'est avéré être très riche en apport de connaissance et de méthodes de raisonnement.

[25] http://scikit-learn.org

7. Bibliographie

1. Docker: https://www.docker.com/
2. Dockerfile: https://docs.docker.com/engine/reference/builder/
3. API plugin Docker: https://docs.docker.com/engine/extend/plugin_api/
4. Git: https://git-scm.com/
5. NFS: https://en.wikipedia.org/wiki/Network_File_System
6. DevOps: https://en.wikipedia.org/wiki/DevOps
7. Job Scheduler: https://en.wikipedia.org/wiki/Job_scheduler
8. Prometheus (métriques): https://prometheus.io/
9. Grafana (affichage de métriques): https://grafana.com/
10. Weka: https://www.cs.waikato.ac.nz/ml/weka/
11. Auto-sklearn: https://github.com/automl/auto-sklearn
12. OpenML: https://www.openml.org/
13. Scikit-Learn: http://scikit-learn.org
14. Feature model: https://en.wikipedia.org/wiki/Feature_model
15. Len Bass, Ingo Weber, Liming Zhu. *DevOps: A Software Architect's Perspective*
16. Michael Httermann. *DevOps for Developers*
17. Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, Frank Hutter. Efficient and Robust Automated Machine Learning
18. *Abhishek Thakur. Approaching (Almost) Any Machine Learning Problem*
19. Alvaro Fernando Lara. *Continuous Integration for ML Projects*
20. Boris Tvaroska. *Using a DevOps Pipeline for a Machine Learning Project*
21. Jeux de données tests: https://archive.ics.uci.edu/ml/datasets.html
*22. Jeux de données tests d'*OpenML*100:* https://www.openml.org/s/14/data
23. Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Thomas Lindauer, Yuri Malitsky, Alexandre Frechette, Holger H Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. benchmark library for algorithm selection. Artif. Intell., 237:41–58, 2016
24. Erwan Brottier, Franck Fleurey, Jim Steel, Benoit Baudry, and Yves Le Traon. Metamodel-based test generation for model transformations: An algorithm and a tool. In Proceedings - International Symposium on Software Reliability Engineering, ISSRE, 2006
25. Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning. Advances in Neural Information Processing Systems 28, 2015.
26. Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys. Towards automatic composition of multicomponent predictive systems. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2016.
27. John R. Rice. The Algorithm Selection Problem. Advances in Computers, 15(C):65–118, 1976.
28. Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. ACM SIGKDD Explorations Newsletter, 2013.
29. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1997.

Annexes

A. Encapsulation avec Docker

Docker est une technologie de conteneurisation. Cette technologie permet de créer et d'exécuter des conteneurs d'applications définis sous la forme d'image. La figure ci-dessous présente un exemple d'image Docker.

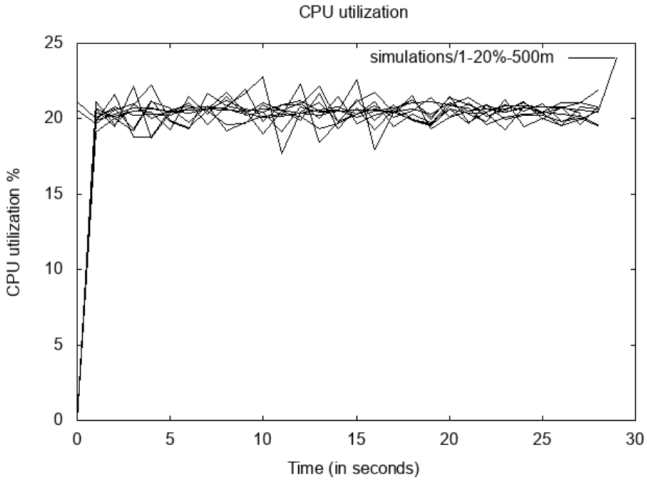
```
# On hérite d'une image de base
FROM rockflows/process-executor:java
# On Ajoute notre binaire à l'image
COPY my-classifier.jar app.jar
# On définit le point d'entrée du conteneur
ENTRYPOINT ["java", "-jar", "app.jar"]
```

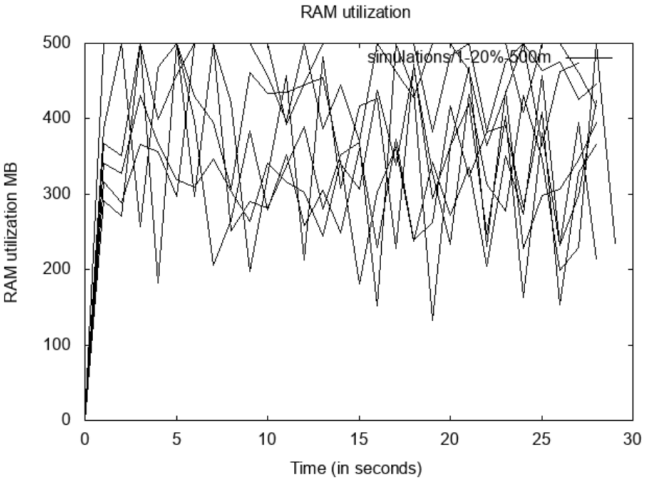
Un exemple de définition d'image Docker.

Cette technologie permet ainsi d'exécuter des applications sans avoir à se soucier des dépendances d'exécutions. C'est grâce à cette fonctionnalité que nous pouvons exécuter des *workflows* de différents langages de programmation. En plus de cette capacité, Docker permet de contrôler l'exécution des conteneurs en terme de ressources système. Un benchmark testant cette capacité d'isolation nous a permis de confirmer le choix de cette technologie. Nous avons étudié deux solutions possibles quant-à l'isolation des ressources: une virtualisation de chaque exécution, ou une conteneurisation. La virtualisation permet un contrôle garantie des ressources mais est gourmande en temps d'instanciation. Les figures ci-dessous présente certains résultats de notre benchmark. Ce benchmark utilise une image Docker permettant d'utiliser toute les ressources disponibles au sein d'un conteneur. Nous avons lancé 10 exécutions par scénario[26] et nous cherchons à vérifier que les conteneurs Docker respectent bien les limites fournis en paramètres en observant les ressources utilisées par l'hôte lors d'une exécution de 30 secondes .

[26] Un scénario correspond à une limitation CPU et ram d'un conteneur

20% CPU - 500MB



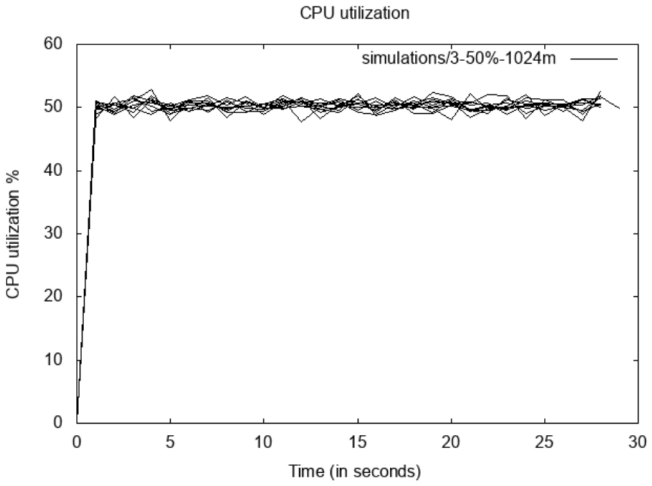


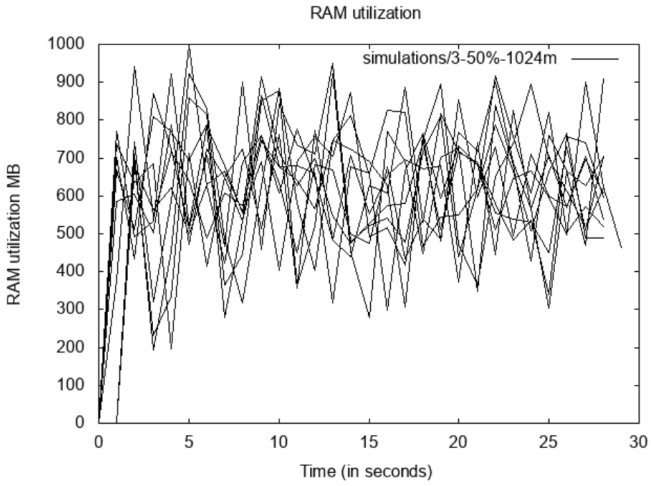
CPU 20% **+ 4%**

RAM <= 500MB

Scénario 20% d'un CPU et 500 Méga-Octets de RAM.

50% CPU - 1024MB



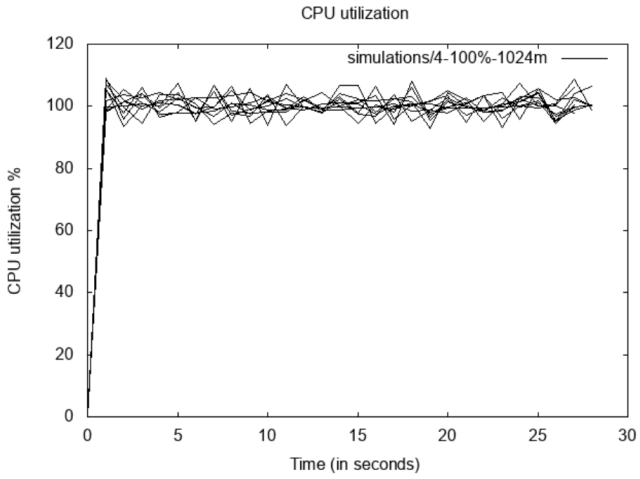


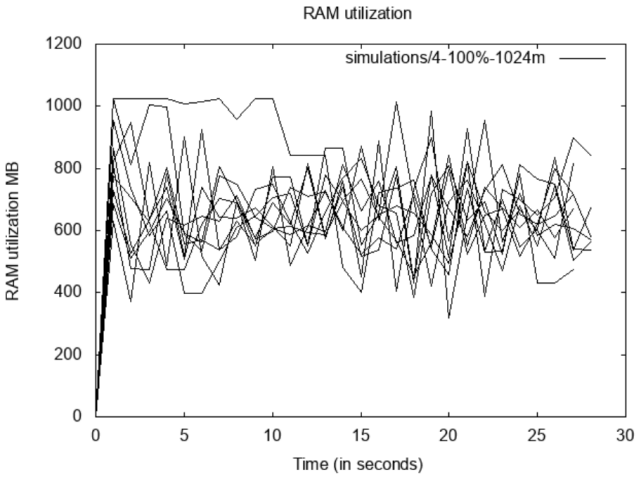
CPU 50% **+ 3%**

RAM <= 1024MB

Scénario 50% d'un CPU et 1024 Méga-Octets de RAM.

100% CPU - 1024MB





CPU 100% **+ 10%**

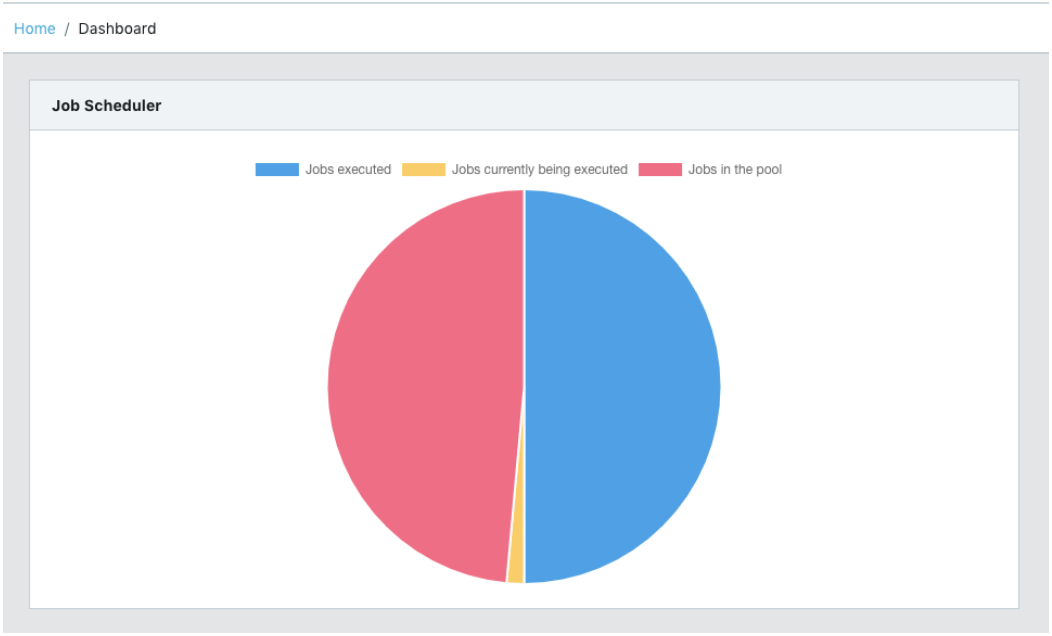
RAM <= 1024MB

Scénario 100% d'un CPU et 1024 Méga-Octets de RAM.

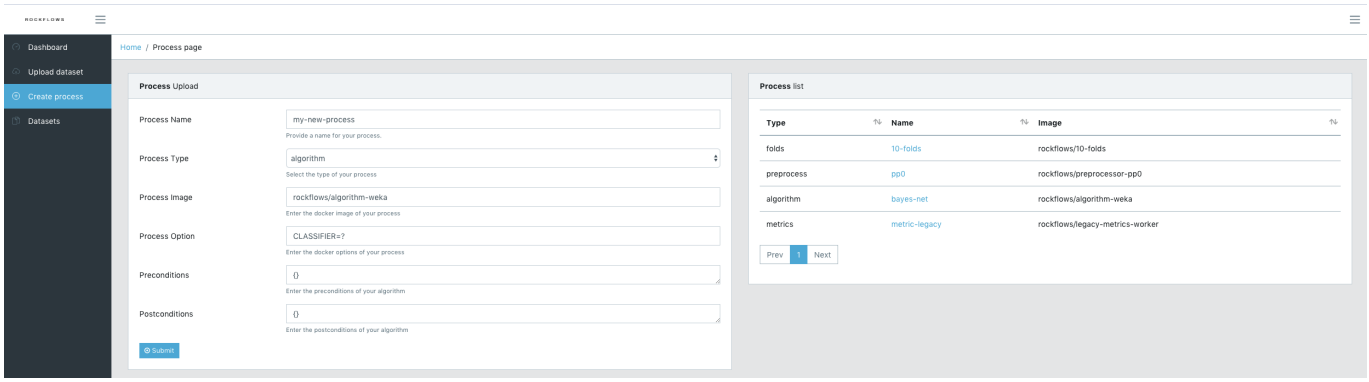
Ce benchmark nous a permis de montrer un biais de 10% d'utilisation du CPU. Cela veut dire que la capacité d'encapsulation des ressources Docker ne permet pas un contrôle absolue des ressources CPU, mais ce biais de 10% n'est pas un problème pour les expérimentations. Il suffit de garder une marge de 10% en plus par conteneur. La restriction sur la mémoire virtuelle est bien respectée.
C'est grâce à ce benchmark que nous avons pu décider d'utiliser la technologie Docker à la place de la virtualisation pour l'exécution des expérimentations. Le biais de 10% est beaucoup moins impactant que le grand temps d'instanciation d'une approche par virtualisation.

B. Interface graphique d'expérimentations

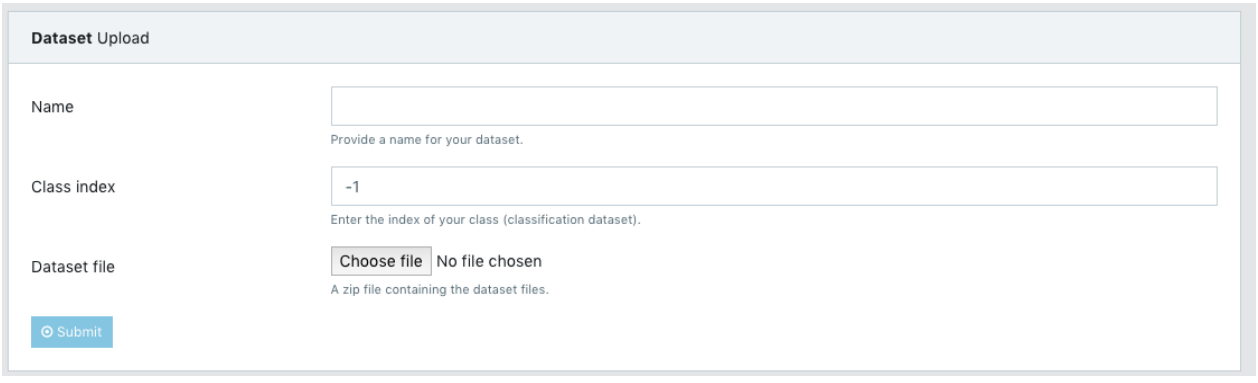
Pour visualiser le suivi des expérimentations, j'ai développé une interface web minimaliste regroupant différentes métriques. Sur cette interface, nous pouvons y trouver l'état du planificateur de tâches (ex. nombre de workflows en cours d'exécution ou en attente), la liste des différents algorithmes ainsi que des jeux de données. De plus, pour chaque jeu de données, une liste des résultats de chaque *workflow* exécuté pour ce dernier est disponible. Les figures ci-dessous montrent ces différentes fonctionnalités.



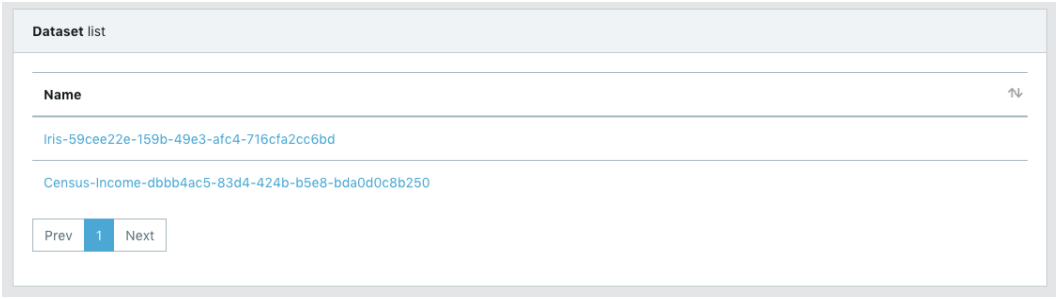
État du planificateur de tâches sous forme de diagramme.



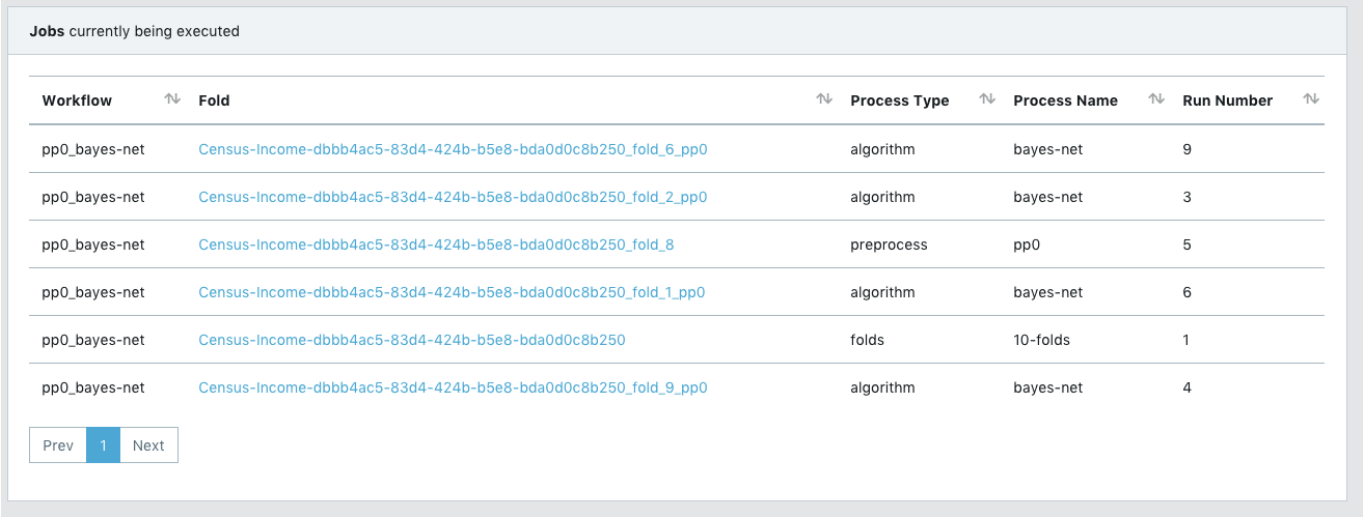
Interface d'ajout d'un algorithme.



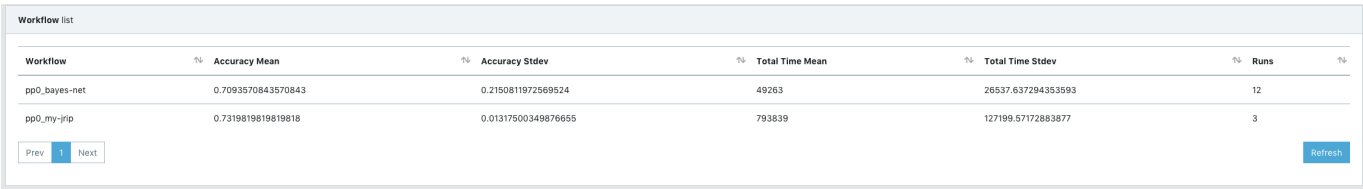
Interface d'ajout d'un jeu de données.



Liste des jeux de données.



Suivi des workflows en cours d'exécution.



Suivi des résultats des exécutions des workflows d'un jeu de données.

Abstract

ROCKFlows is a project aimed to simplify the creation of machine learning workflows. Its goal is to produce, for a given problem, the best workflow from a portfolio. In order to build this portfolio, experimentations must be run. These experimentations are used to build predictive models for a given problem and dataset. ROCKFlows had an experiment system but this latest had to evolve in order to respond to new constraints. With a new architecture, my apprenticeship made the platform evolve to be able to handle a lot of experimentations, to automate different tasks such as the composition of valid workflows, and opened perspectives such as the ability for the platform to bootstrap itself.