Dominique Piché, Samuel Schwab, Vincent Leduc

Rapport TP1

Présenté à

Marios Fokaefs, Ph.D

INF8371

Ingénierie de la qualité en logicielle

21 février 2019

École Polytechnique Montréal

Weka - Plan d’assurance

[**Introduction**](#_qrrsp72vkaij) **3**

[**Critères de qualité**](#_cwl595775hqr) **4**

[Fonctionnalité](#_qo9xbkptl801) 4

[Fiabilité](#_b3bk20k71dhu) 4

[Maintenabilité](#_doqzuuh0fpgu) 4

[**Objectifs et mesures de validation**](#_h6czpfqj9b9l) **5**

[Fonctionnalité](#_1sm3i91kv7ev) 5

[Exactitude](#_76tmtne9mobl) 5

[Précision](#_kcwqd8xpk4b1) 5

[Fiabilité](#_d9yxu8ath1bl) 5

[Récupérabilité](#_yu1ce6cgxw1q) 5

[Maintenabilité](#_xedy3mrhig33) 5

[Analysabilité](#_pvea13162lzb) 5

[Testabilité](#_t82rsn8qcdg) 5

[**Stratégie de validation**](#_dblcv5p019vy) **6**

[**Plan d’intégration continue**](#_c7za7ly3kdd6) **8**

[**Ajout d’un nouvel algorithme**](#_9ea7ypl14rcf) **9**

[**Vidéo de la pipeline Travis**](#_t9zoplwfyom2) **9**

## 

# Introduction

une introduction courte qui spécifie le type du système logiciel, l’importance de sa qualité, les parties pertinentes

Weka est un système logiciel *open source* regroupant plusieurs algorithmes d’apprentissage machine et d’exploration de données. Ces algorithmes concernent la préparation des données, leur classification, leur partitionnement, leur visualisation, l’exploration par association et l’analyse par régression[[1]](#footnote-0). Ce système se veut être une librairie pour des applications utilisant le *big data* et l’apprentissage profond. Comme le logiciel est conçu pour être utilisé et développé par un grand nombre de personne, il est essentiel que sa qualité soit assurée. D’ailleurs, plusieurs entreprises pourraient être affectées par de quelconques défaillances, ce qui pourrait engendrer des pertes économiques directes et indirectes considérables. Les cinq fonctionnalités retenues pour évaluation sont les suivantes:

1. Farthest First[[2]](#footnote-1): Algorithme de partitionnement de données minimisant la distance de celles-ci dans un groupe donné et maximisant la distance entre les différents groupes de données.
2. Simple K Means[[3]](#footnote-2): Algorithme de partitionnement de données regroupant les données en *k* groupes, où chaque donnée est associée à la partition dont la moyenne est la plus proche de sa valeur.
3. Poisson Estimator[[4]](#footnote-3): Algorithme permettant de déterminer, à l’aide d’une distribution de Poisson, une fonction pouvant estimer les valeurs de la série de données.
4. Normal Estimator[[5]](#footnote-4): Algorithme permettant de déterminer, à l’aide d’une distribution normale, une fonction pouvant estimer les valeurs de la série de données.
5. FP Growth[[6]](#footnote-5): Règle d’association permettant de compresser un ensemble de données sous forme d’arbre, découplant l’exploration de données des bases de données.

# Critères de qualité

Les trois critères de qualités qui sont imposés dans le cadre de ce TP sont les suivants :

* Fonctionnalité
* Fiabilité
* Maintenabilité

Pour chacun des critères nous avons établis la liste des sous-critères qui étaient les plus importants pour ce projet.

## Fonctionnalité

L’évaluation de la qualité au niveau de la fonctionnalité d’un logiciel correspond à la capacité du logiciel d’accomplir les tâches pour lesquels il a été conçu.

Les sous-critères sont les suivants :

* Exactitude: L’exactitude d’un système logiciel correspond à sa capacité à donner des réponses justes aux demandes qui lui sont faites.
* Précision: La précision d’un système logiciel correspond à sa capacité à donner des réponses dont l’incertitude sur les réponses est minimale.

## Fiabilité

L’évaluation de la qualité au niveau de la fiabilité correspond à l’évaluation du comportement du système logiciel en cas de problème.

Les sous-critères sont les suivants :

* Récupérabilité: La récupérabilité correspond à la capacité d’un système logiciel à retrouver un comportement adéquat suite à une faille du système.

## Maintenabilité

L’évaluation de la qualité au niveau de la maintenabilité correspond à l’analyse des qualités du code source facilitant l’entretien et le développement du produit logiciel dans le temps.

Les sous-critères sont les suivants :

* Analysabilité: L’analysabilité d’un système logiciel correspond à sa facilité de compréhension et à l’encapsulation de ses fonctionnalités.
* Testabilité: La testabilité d’un système logiciel correspond à sa facilité de production et d’exécution de tests sur ses différentes fonctionnalités et composantes.

# Objectifs et mesures de validation

L’objectif de ce TP est de valider la qualité d’une version limitée du logiciel weka selon les différents critères et sous-critères identifiés dans la section précédente. Pour mesurer si le logiciel respecte ces critères, il nous faut établir des mesures de validation

## Fonctionnalité

#### Exactitude

Chaque réponse donnée par le logiciel doit être la bonne (taux de réussite de 100%)

#### Précision

Lorsque le logiciel fournit des réponses numériques, en plus d’être justes, celles-ci doivent être précises à 99,999% par rapport aux solutions mathématiques parfaites.

## Fiabilité

#### Récupérabilité

90% du temps, lorsque le programme rencontre une défaillance, les données ne sont pas corrompues et sont récupérables à 100%

## Maintenabilité

#### Analysabilité

Le logiciel doit avoir un degré de couplage de classes inférieur à 5 100% du temps.

#### Testabilité

Lorsqu’une nouvelle classe ou fonctionnalité est ajoutée et que sa classe de tests est ajoutée, il faut que cette classe de test dispose de tests unitaires générés automatiquement.

# 

# Stratégie de validation

Pour les algorithmes de partitionnement (FarthestFirst et Simple K Means), les classes de test héritent de la classe AbstractClustererTest, qui contiennent les tests suivants :

Portée des tests:

Les tests unitaires ont comme portée une seule classe. Les tests de régression ont comme portée le logiciel au complet.

Environnement de test:

Les tests sont roulés dans un environnement linux, à l’aide du package maven. les tests sont écrits avec le package java junit. Ils sont exécutés automatiquement lorsque le code source est modifié dans le git. Si les tests ne passent pas, le code modifié est rejeté.

Détails des tests:

* Exactitude: Les tests sont roulés avec les valeurs attendues des données, notamment en ce qui concerne les estimateurs et les fonctions rapportées. L’objectif est de s’assurer que le système retourne des valeurs justes.
* Précision: Les tests sont roulés et comparés avec les valeurs attendues des données. L’objectif est de s’assurer que l’incertitude sur les résultats est minimale.
* Récupérabilité: Les tests sont roulés au lancement du programme suite à un arrêt forcé. La cohérence des données récupérées par rapport à celles présentes avant l’arrêt est évaluée. L’objectif est de mesurer que le comportement logiciel est adéquat en cas de panne.
* Analysabilité: Les tests de coupling sont exécutés sur le build le plus récent de la branche *master*. La valeur obtenue est comparée avec celle du seuil établi. L’objectif de ce test est d’assurer une forte cohésion interne entre les classes et une faible interdépendance afin d’assurer la facilité de compréhension de l’ensemble du système.
* Testabilité: Le système vérifie à chaque ajout de fonctionnalité (commit) la présence de tests liés aux classes en question. Une mise en garde est levé dans le cas où cette clause n’est pas respectée. L’objectif de ce test est de s’assurer que toute fonctionnalité est testée et développée avec cette fin en perspective.

Processus des tests

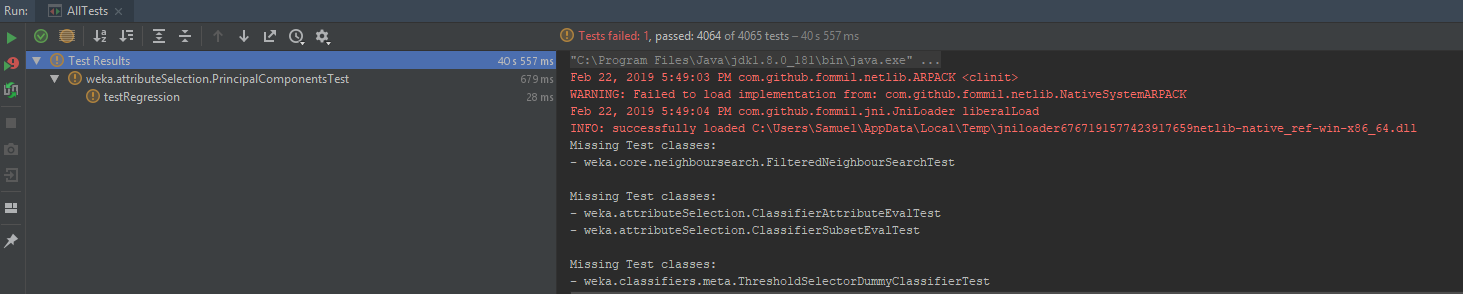
Les test qui sont roulés vont lire les données dans le fichier .ref contenu dans le dossier correspondant dans le package wekarefs.weka. ces données sont les résultats attendus des tests roulés par les classes de test correspondantes

Programme des tests:

Les tests seront exécutés à chaque fois qu’une requête de modification du code source de l’entrepôt central est déposée. Le pipeline Travis exécute les tests, et si un test échoue, la modification est rejetée.

Rapport des tests

Tous les tests de weka 3.8 ont été roulés à partir de Windows 10 avec le jdk 1.8.0\_181 dans l’IDE intelliJ 2018.2.3. Un test a échoué, et 4064 ont passé.



La stratégie pour identifier la raison du test échoué serait de rouler les tests dans d’autres environnements et de vérifier si le fichier .ref contenant les résultats attendus du test sont valides.

# 

# Plan d’intégration continue

L’outil choisi pour l’intégration continue et pour le déploiement continu est Travis CI.La configuration du pipeline doit être inscrite dans un fichier nommé .travis.yml situé à la racine du répertoire git.

Le pipeline Travis va contenir trois étapes :

* Compilation
* Test
* Déploiement

Pour la première étape, on veut tout simplement rouler la commande *mvn compile*.



Ensuite, pour l’étape de test, la commande *mvn test* va trouver et exécuter tous les fichiers de test contenus dans le projet. Ceci inclut les tests spécifiés à la question précédente.



Finalement, pour l’étape de déploiement, on utilise l’option *deploy* de Travis CI pour déployer le code source dans la section Releases du répertoire GitHub. On pourrait éventuellement y déployer des installateurs pour weka pour Windows ou macOs avec l’option *file.* Pour ne pas déployer un nombre de fois trop élevé, on limite le déploiement aux tags, afin de pouvoir déployer uniquement des versions stables sur notre page GitHub. Aussi, afin de donner accès à GitHub au pipeline Travis, on utilise l’option api\_key qui contient un token encrypté avec les permissions nécessaires.



# 

# Ajout d’un nouvel algorithme

L’algorithme qu’on ajoute est l’algorithme d’association FilteredAssociation. Étant donné que cet algorithme hérite de la classe SingleAssociatorEnhancer qui elle même hérite de la classe AbstractAssociator, il est facile de garantir la qualité du système. En effet, il est possible de valider que notre nouvel algorithme remplit bien toutes les responsabilités d’un AbstractAssociator en créant une classe de test qui hérite de AbstractAssociatorTest. Cette nouvelle classe de test (FilteredAssociatorTest) sera donc détectée automatiquement par le script de test de maven (mvn test) et une suite de test sera facilement générée et retournée par la méthode suite()



Cette suite de test fera automatiquement des test unitaires pour tester le nouvel algorithme et des tests de régression pour valider que la nouvelle classe s'intègre bien au système. Il n’y a donc pas de besoin de modifier le plan de test, sauf pour indiquer la nouvelle classe de test qui doit être créée.

# Vidéo de la pipeline Travis

<https://www.youtube.com/watch?time_continue=22&v=5bmIyI8f-5I>

1. <https://www.cs.waikato.ac.nz/ml/weka/> [↑](#footnote-ref-0)
2. <https://github.com/Waikato/weka-3.8/blob/master/weka/src/main/java/weka/clusterers/FarthestFirst.java> [↑](#footnote-ref-1)
3. <https://github.com/Waikato/weka-3.8/blob/master/weka/src/main/java/weka/clusterers/SimpleKMeans.java> [↑](#footnote-ref-2)
4. <https://github.com/Waikato/weka-3.8/blob/master/weka/src/main/java/weka/estimators/PoissonEstimator.java> [↑](#footnote-ref-3)
5. <https://github.com/Waikato/weka-3.8/blob/master/weka/src/main/java/weka/estimators/NormalEstimator.java> [↑](#footnote-ref-4)
6. <https://github.com/Waikato/weka-3.8/blob/master/weka/src/main/java/weka/associations/FPGrowth.java> [↑](#footnote-ref-5)