VINCENT CLERIS – 1794655
THOAI-VI LE – 1794802
GABRIEL HÉLIE - 1791764
JULIEN GAUTHIER - 1791257
GABRIEL CAMPBELL - 1761276

LOG8371 - Ingénierie de la qualité en logiciel

TP2 : Efficacité et performance

Remis à
Marios Fokaefs

École Polytechnique de Montréal
21 mars 2019

Table des matières

1. Introduction **2**
1.1 Description du projet 2
1.2 Importance de la qualité 2
1.3 Portée du document 2

2. Fonctionnalités ciblées **2**
2.1 Canopy Clustering (Clustering) 2
2.2 Hierarchical Clustering (Clustering) 3
2.3 Random Forest (Classifying) 3
2.4 AdaBoost or Adaptive Boosting (Classifying) 3
2.5 Apriori Algorithm (Associating) 3
2.6 Bubble Sorting 3

3. Objectifs de Qualité **3**
3.1 Fonctionnalité 4
3.2 Fiabilité 4
3.3 Maintenabilité 4
3.4 Efficacité et performance 4
3.5 Sous-critères et objectifs mesurables 4

4. Activités et stratégies de validation **5**
4.1 Normes de codage 5
4.2 Revue de code et Pull Requests 5
4.3 Stratégie de test 5

5. Activités de tests **6**
5.1 Cas de tests 6
5.1.1. Tests de fonction 6
5.1.2 Tests de régression 6
5.1.3 Tests d'intégration 6
5.1.4 Tests de performance (ou de charge) 7
5.1.4 Tests de stress 7
5.2 Exemple de suite de tests 7

6. Plan d'intégration continue **8**

7. Médiagraphie **9**

1. Introduction

1.1 Description du projet

Le logiciel Weka 3 est un logiciel contenant une banque d'algorithmes pour effectuer des travaux relatifs à l'exploration de données (*DataMining*). Le logiciel contient des outils pour visualiser et manipuler des ensembles de données, préparer les données et encore plus comme effectuer des grappes de données et les classifier selon des attributs. Le logiciel a été développé en Java et est en code source libre. C'est à dire qu'il est disponible gratuitement pour tous et qu'il est possible pour un utilisateur de modifier le logiciel pour l'améliorer selon ses besoins. [1]

1.2 Importance de la qualité

On souhaite que Weka soit un logiciel de qualité parce qu'il sert de composante de base à différents projets d'intelligence artificielle. Il est ainsi un élément important pour un système qui l'utilise. La collection d'algorithmes, d'outils et de fonctionnalités qu'il propose ne sont intéressants que s'ils sont performants, fiables et maintenables pour un projet qui veut les utiliser. Il est important de noter qu'un logiciel qui se colle à des standards de qualité élevés est sujet, la majorité du temps, à une introduction réduite de bugs et par le fait même à la réduction des coûts de développement et de maintenance logiciel.

1.3 Portée du document

Le présent document a pour objectif de documenter les différentes pratiques relatives à la maintenabilité d'un niveau de qualité pour le logiciel Weka 3. Il portera d'abord sur les fonctionnalités de Weka qui nous intéressent dans le cadre de ce travail et les objectifs de qualité déterminés pour le projet. Il portera ensuite sur les différents moyens employés pour conserver la qualité du logiciel à un niveau respectable. Il abordera les différents tests effectués par l'équipe pour vérifier que le logiciel fonctionne toujours comme souhaité après chaque modification du code source, puis finalement sur les méthodes utilisées pour le déploiement automatique des changements apportés et pour le contrôle de versions.

2. Fonctionnalités ciblées

2.1 Canopy Clustering *(Clustering)*

L'algorithme de clustering canopy est un algorithme de prétraitement pour permettre à d'autres algorithmes de clustering d'effectuer le travail plus facilement et rapidement sur des jeux de données qui n'auraient pas pu être traités efficacement, dû par exemple à une quantité de données trop élevée. Il est souvent utilisé par exemple en prétraitement pour les algorithmes de clustering hiérarchique et k-moyenne. [3]

2.2 Hierarchical Clustering *(Clustering)*

En analyse des données, la notion de *hierarchical clustering* est une méthode d'analyse de grappe (*clusters*) populaire qui cherche à créer une hiérarchie de grappes. Pour ce faire, la méthode créée des clusters avec un ordre prédéterminé soit de haut en bas (*top-down*) ou de bas en haut (*bottom-top*). L'approche *top-down* est appelée "*divisive"* et l'approche *bottom-up* est appelée "*agglomerative"*. [4]

2.3 Random Forest *(Classifying)*

Random Forest, aussi appelé "forêts d'arbres décisionnels", est un des algorithmes les plus utilisés dans le domaine de l'apprentissage automatique. Comme son nom l'indique, cet algorithme construit des arbres de décision afin de créer des modèles prédictifs pour les problèmes de classification et de régression. Pour ce faire, le modèle créé une forêt d'arbre de décision aléatoire. Les arbres ne doivent pas être corrélés entre eux ou similaires parce que c'est l'addition des prédictions de chacun des arbres qui formule une meilleure prédiction. Pour ce faire, on doit utiliser un algorithme de *bagging* préalablement sur le jeu de données. [5]

2.4 AdaBoost *or Adaptive Boosting (Classifying)*

L'*AdaBoosting* est un méta-algorithme ayant pour objectif d'entraîner des *week learners* et modifier des coefficients pour que ceux-ci réagissent mieux la prochaine fois qu'ils seront mis à l'épreuve. Cet algorithme est adaptif, d'où son nom. [6]

2.5 Apriori Algorithm *(Associating)*

L'algorithm *Apriori* permet d'associer des items à des ensembles de données un à un et tant et aussi longtemps que l'item qui veut être ajouté fait du sens dans l'ensemble de données actuels. Plus l'ensemble devient gros, plus il devient difficile d'ajouter un item. L'algorithme se termine lorsqu'il est impossible d'associer de nouveaux items aux ensembles de données. [7]

2.6 Bubble Sorting

L'algorithm *Bubble Sort* permet tirer les différents éléments d'un jeu de données. Pour ce faire il effectue des permutation entre un élément du jeu de donnée et le prochain élément en parcourant les données de façons linéaire. Le processus est répétés jusqu'à ce qu'il n'y ait plus aucune permutation possible et à ce moment là, le jeu de donnée est complètement trié.

3. Objectifs de Qualité

Les critères de qualité importants pour le projet Weka sont la *fonctionnalité*, la *fiabilité* et la *maintenabilité* du projet. Ces critères sont ciblés parce qu'ils sont, comme indiqués au point

1.2, indispensables pour que le logiciel soit intéressant pour les utilisateurs. Voyons à quoi correspondent ces critères de qualité.

3.1 Fonctionnalité

La qualité de fonctionnalité est la capacité du logiciel à répondre aux exigences du client et à produire le résultat attendu.

3.2 Fiabilité

La fiabilité assure qu'un logiciel informatique fonctionne de manière cohérente selon ses spécifications en tout temps.

3.3 Maintenabilité

La maintenabilité logicielle correspond au niveau d'efficacité et de facilité pour un système à être modifié, amélioré ou changé par un développeur qui souhaite ajouter une fonctionnalité, en améliorer une autre ou transformer le logiciel.

3.4 Efficacité et performance

La performance d'un logiciel se mesure par le rapport entre la quantité de ressources utilisées et la quantité de résultats livrés. En d'autres mots, il s'agit de la quantité de ressources utilisées selon des conditions précises. On dira également qu'un logiciel est performant s'il livre les résultats attendus dans des délais raisonnables.

3.5 Sous-critères et objectifs mesurables

Voici les différents sous-critères rattachés à ces critères importants ainsi que des objectifs mesurables qui permettent de définir si ces critères de qualité sont atteints ou non pour le logiciel.

Critères Primaires	**Sous-critères**	**Objectifs mesurables**
Fonctionnalité	Exactitude	Les algorithmes agissent de façon attendue dans plus de 95% des cas.
	Complétude	Le logiciel propose des algorithmes pour effectuer de l'exploration de données efficace sur plus de 95% des ensembles de données.
Fiabilité	Tolérance aux fautes	Le logiciel continu de fonctionner correctement et ne corrompt pas les données si jamais une erreur logicielle survient.
Maintenabilité	Testabilité	Chaque algorithme et/ou fonctionnalité du

		logiciel doit être testable individuellement (tests unitaires).
	Analysabilité	Le code source doit suivre les normes de codage de l'entreprise et être commenté pour faciliter son analyse. Un analyste doit pouvoir bien comprendre le code après une première lecture.
Efficacité et performance	Comportement temporel	Le programme doit pouvoir effectuer chacun des algorithmes sur un jeu de données d'une centaine de données en moins de 5 secondes.
	Utilisation de ressources	Le programme ne doit pas utiliser plus de 80% du CPU des container déployé.
	Capacité	Le déploiement du programme doit pouvoir conserver l'utilisation du CPU des machines de déploiement à un niveau acceptable en utilisant la stratégie de *scaling*.

4. Activités et stratégies de validation

4.1 Normes de codage

Chaque classe doit être commentée à l'aide d'un en-tête pour indiquer qu'elle appartient au logiciel Weka et doit contenir une courte description de l'objectif de la classe. Les méthodes doivent également être commentées et avoir une description associée. Les nouvelles fonctionnalités doivent être accompagnées de tests unitaires.

4.2 Revue de code et *Pull Requests*

Chaque modification au code source doit faire l'objet d'un *pull request* et être revue puis acceptée par un responsable du logiciel. Cette pratique oblige les développeurs à respecter les normes de codage du projet et permet de s'assurer que les modifications apportées au logiciel n'affectent pas le bon fonctionnement du code.

4.3 Stratégie de test

La stratégie choisie pour tester le logiciel est la stratégie de test bottom-up. Elle consiste à tester individuellement les petits modules de l'application puis des ensembles de modules de plus en plus gros jusqu'à tester l'ensemble du projet à l'aide de tests d'intégrations.

5. Activités de tests

Les tests seront effectués après chaque modification au code source afin de vérifier que l'état du logiciel n'est pas endommagé. Ils seront effectués automatiquement grâce à l'intégration continue des modifications apportées. L'intégration continue est assurée par Travis CI.

5.1 Cas de tests

Voici les différents types de tests qui sont effectués à chaque intégration de nouveau code pour vérifier l'état du système. [8]

5.1.1. Tests de fonction

Objectif de test:	Assurer et vérifier le bon fonctionnement des algorithmes d'exploration de données individuellement.
Technique:	Exécution des tests unitaires pour chacun des algorithmes en utilisant un ensemble de données de base.
Critère de complétion:	- Les tests planifiés ont tous été exécutés - Les défauts trouvés en cas d'erreurs ont été corrigés
Considérations spéciales:	Aucune

5.1.2 Tests de régression

Objectif de test:	Assurer que l'ajout d'une fonctionnalité ou d'un algorithme ne perturbe pas le bon fonctionnement des autres algorithmes sous une version antérieure.
Technique:	Exécution des tests unitaires et des cas d'utilisation pour tous les algorithmes qui n'ont pas été modifiés.
Critère de complétion:	- Les tests planifiés ont tous été exécutés - Les défauts trouvés en cas d'erreurs ont été corrigés
Considérations spéciales:	Aucune

5.1.3 Tests d'intégration

Objectif de test:	Assurer qu'il n'y a pas de nouveaux défauts lorsque les différentes composantes du système interagissent ensemble.

Technique:	Exécution des tests en boîte noire et des tests en boîte blanche.
Critère de complétion:	- Les tests planifiés ont tous été exécutés - Les défauts trouvés en cas d'erreurs ont été corrigés
Considérations spéciales:	Aucune

5.1.4 Tests de performance (ou de charge)

Objectif de test:	Assurer que le temps de réponse du logiciel demeure adéquat selon son degré de sollicitation.
Technique:	Exécution des tests de charge pour plusieurs scénarios de cas de charge : charges réduite, moyenne, augmentée et augmentée exceptionnelle.
Critère de complétion:	- Les tests planifiés ont tous été exécutés - Les défauts trouvés en cas d'erreurs ont été corrigés
Considérations spéciales:	Utilisation de l'outil JMeter.

5.1.4 Tests de stress

Objectif de test:	Assurer que le comportement du logiciel demeure exact lors de l'utilisation de ressources partagées et rares.
Technique:	Surcharge de la mémoire vive et du disque dur lors de l'exécution du logiciel.
Critère de complétion:	- Les tests planifiés ont tous été exécutés - Les défauts trouvés en cas d'erreurs ont été corrigés
Considérations spéciales:	Aucun test écrit.

5.2 Exemple de suite de tests

À la suite d'un ajout au code source pour un nouvel algorithme par exemple, une suite de tests typique pour les fonctionnalités choisies serait d'effectuer les tests unitaires du nouvel

algorithme puis de faire ensuite les tests de régression pour s'assurer que les autres algorithmes n'ont pas été affectés par le changement. On aurait alors les tests unitaires pour le nouvel algorithme, le "*Canopy Clustering"*, le "*Hierarchical Clustering"*, le "*Random Forest"*, "*l'AdaBoost"*, "*l'Aprirori Algorithm"* ainsi que le *"Bubble Sorting"*. Finalement, même si chacun des algorithmes fonctionne correctement individuellement, il faut effectuer des tests d'intégration afin de confirmer qu'ils puissent s'effectuer les uns après les autres de façon fiable. On fera alors quelques tests de cas d'utilisation utilisant les fonctionnalités disponibles par le logiciel.

6. Plan d'intégration continue

Afin d'assurer l'intégration continue de modification au code source, nous utilisons le logiciel Travis CI. Travis CI est intégré au repo Git et s'occupera de lancer toutes les opérations nécessaires à l'intégration de nouveau code automatiquement. Dans notre cas, Travis CI s'occupera de bien démarrer chaque test pertinent après un *push* sur le repo Git. Cette méthode est efficace parce qu'un segment de code qui souhaite être *pushed* sur le repo doit avoir été revu lors d'un *pull request* et une fois accepté, Travis CI s'assurera qu'aucun nouveau bug est introduit dans le système grâce aux activités de tests.

Il est important de noter qu'en cas d'intégration d'un nouvel algorithme, il faudra mettre à jour la suite de tests qui sera lancée par Travis CI, puisqu'on voudra utiliser les tests unitaires de cet algorithme pour les tests de régression et qu'on voudra également l'ajouter aux tests d'intégration.

7. Médiagraphie

[1] Weka 3: Data Mining Software in Java. [En ligne]. Disponible : https://www.cs.waikato.ac.nz/~ml/weka/index.html

[2] Assurance qualité logicielle. [En ligne]. Disponbile : https://fr.wikipedia.org/wiki/Assurance_qualit%C3%A9_logicielle

[3] Canopy clustering algorithm. [En ligne]. Disponible : https://en.wikipedia.org/wiki/Canopy_clustering_algorithm

[4] Hierarchical clustering. [En ligne]. Disponible : https://en.wikipedia.org/wiki/Hierarchical_clustering

[5] Random forest. [En ligne]. Disponible : https://en.wikipedia.org/wiki/Random_forest

[6] Adaboost. [En ligne]. Disponible : https://en.wikipedia.org/wiki/AdaBoost

[7] Apriori algorithm. [En ligne]. Disponible : https://en.wikipedia.org/wiki/Apriori_algorithm

[8] Olivier gendreau, gabarit de Plan de tests logiciel. [En ligne]. Disponible : https://moodle.polymtl.ca/pluginfile.php/88637/mod_resource/content/4/Plan%20de%20tests%20logiciels.docx