École Polytechnique de Montréal

Département de génie informatique et logiciel

LOG8371
Ingénierie de la qualité en logiciel

TP2- Efficacité et Performance

Soumis à Mohamed-Essaddik Benyahia

Soumis par
Julien Arès (1802992)
Olivier Filippelli (1775057)
Kim Thuyen Ton (1789211)
Alex Tran (1799594)

Groupe 03 – Équipe Bravo
Hiver 2019

29 mars 2019

Table des matières

Les nouvelles parties du plan sont démarquées par des lignes pointillées.

Introduction

Le plan d'assurance se porte sur le logiciel Weka. Ce logiciel est open-source. Il est écrit par l'université de Waikato en Nouvelle-Zélande. C'est un outil servant au data mining. Il contient toutes les composantes pour pouvoir utiliser plusieurs algorithmes type de data mining tels que le clustering et la classification et visualiser les données et les résultats. Ce plan se concentre sur deux algorithmes de classification, deux de clustering et un d'association: l'algorithme de bayes naïf, l'algorithme de régression linéaire, l'algorithme de k-mean, l'algorithme de clustering hiérarchique et l'algorithme apriori respectivement.

Des chercheurs utilisent ce logiciel et comptent sur son bon fonctionnement pour avancer leur recherche. Qu'ils soient des chercheurs en intelligence artificielle ou des chercheurs dans d'autres domaines connexes ayant recours à l'intelligence artificielle comme outil durant leur recherche, il est important pour eux que le logiciel soit de bonne qualité. Il est crucial de bien traiter les données pour obtenir des résultats pertinents pour les chercheurs.

Exigences fonctionnelles

Il est important de comprendre ce que les différents algorithmes font pour pouvoir s'assurer de leur qualité. Sans comprendre le comportement attendu, il est impossible de faire une bonne évaluation de la qualité du logiciel.

Algorithme de Bayes Naïf

Cet algorithme classifie les données selon les différents *features* attachés. Cet algorithme se base sur la prémisse que toutes les *features* sont indépendantes l'une de l'autre.

Algorithme de régression linéaire

Cet algorithme essaye de trouver la meilleure droite pour modéliser un ensemble de données selon la relation entre les variables indépendante et dépendante.

Algorithme de k-means

Cet algorithme regroupe les données parmi k regroupements en les regroupant selon leur distance du centre de chaque *cluster*.

Algorithme de clustering hiérarchique

Cet algorithme sépare chaque donnée dans des regroupements de manière hiérarchique. Il existe plusieurs manière de le faire. L'approche agglomérative part avec toutes les données dans leur propre regroupement et les rassemble progressivement selon le critère de distance choisi. L'approche de division commence avec un seul regroupement contenant toutes les données et sépare une donnée à la fois selon le critère de distance choisi.

Algorithme apriori

Cet algorithme associe les données entre elle selon la fréquence d'apparition des données. Pour qu'une association soit accepté, elle doit apparaître au moins autant de fois qu'un seuil prédéterminé parmi les données.

Algorithme de perceptron à plusieurs couches

Cet algorithme implémente un réseau de neurones avec une couche d'entrée, une couche de sortie et une couche dite cachée qui peut contenir plusieurs couches de noeuds. Chaque

noeud utilise une fonction non-linéaire pour propager l'information à la prochaine couche. Cet algorithme utilise la rétropropagation pour entraîner son modèle. Les erreurs sont calculées à la couche de sortie et retransmises aux couches antérieures.

Critères de qualités

Des critères de qualités sont importants pour bien évaluer la qualité d'un logiciel. Les critères choisis sont la fonctionnalité, la fiabilité, la maintenabilité et l'efficacité de performance. Pour chacun des critères, des sous-critères sont choisi pour mieux définir des objectifs de qualité.

Fonctionnalité

Le critère de fonctionnalité est le critère qui concerne ce que le produit est supposé faire selon les requis prédéterminés. Parmi les sous-critères, les sous-critère de *functional completeness* et de *functional correctness* sont particulièrement importants pour des chercheurs.

Le sous-critère de *functional completeness* vérifie à quel point les fonctions couvrent toutes les tâches déterminées et répondent au besoin des utilisateurs. Ce sous-critère est important, car les chercheurs veulent avoir le plus d'outils à leur disposition pour avancer leur recherche. Plus le logiciel atteint un pourcentage élevé, plus il sera intéressant pour les chercheurs voulant utiliser plusieurs algorithmes.

Le sous-critère de *functional correctness* vérifie à quel point les fonctions retournent les résultats attendus avec la précision voulue. Il est important d'avoir des résultats correctes, car les chercheurs se basent sur elles pour leur recherche. Des mauvais résultats pourraient engendrer une mauvaise conclusion à l'expérience et induire en erreur la communauté scientifique.

Fiabilité

Le critère de fiabilité est le critère qui observe à quel point le logiciel performe dans différentes conditions contrôlées pour bien identifiés le comportement du logiciel. Parmi les sous-critères de *maturity*, de *fault tolerance* et de *recovery* sont importants à tester pour Weka.

Le sous-critère de *maturity* vérifie à quel point le logiciel répond aux besoins des utilisateurs dans des conditions normales d'utilisation. Ces conditions sont les conditions optimales pour le logiciel pour ne pas causer d'erreurs de fonctionnement. Ce critère est important pour les chercheurs, car il faut qu'ils puissent utiliser la version courante en tout temps. Il ne faut pas que la nouvelle version empêche soudainement l'avancement de leur recherche. De plus, il faudrait que le logiciel offre le plus rapidement possible toutes les algorithmes disponibles de data mining selon l'avancement de la littérature scientifique.

Le sous-critère de *fault tolerance* vérifie à quel point le logiciel peut fonctionner malgré la présence d'erreur de logiciel ou de matériel. Ce sous-critère est important puisque les chercheurs s'attendent à avoir des bons résultats sans corruption. Il faudrait que le logiciel invalide les résultats en cas d'erreurs pour ne pas induire les chercheurs en erreur.

Le sous-critère de *recoverability* vérifie à quel point le logiciel peut retourner à un état normal après une interruption grave. Ce sous-critère est important puisque les chercheurs veulent pouvoir utiliser le logiciel après une interruption grave et ne pas devoir à réinstaller le logiciel.

Maintenabilité

Le critère de maintenabilité est le critère qui observe à quel point le logiciel est facilement maintenable par l'équipe de maintenance. Parmi les sous-critères, les sous-critère d'*analysability* et de *modifiability* sont importants à considérer.

Le sous critère d'*analysability* vérifie à quel point il est facile de comprendre les dépendances des différents modules et ainsi facilement prévoir les impacts sur les anciennes fonctionnalités par les changements ou facilement diagnostiquer le logiciel en cas d'erreurs. Cela est important, car plus le logiciel est facilement analyser, plus les itérations comportent moins de risques. Cela permet donc aux nouvelles fonctionnalités d'être implémentées plus rapidement. Cela diminue aussi le risque d'introduire des erreurs et facilite l'identifications des causes de ses derniers.

Le sous-critère de *modifiability* vérifie à quel point le logiciel peut être efficacement modifier sans introduire des erreurs ou dégrader des anciennes fonctionnalités. Cela est important,

car il ne faut pas que les changements de versions brisent le logiciel et empêche les chercheurs de continuer leurs travaux.

Le sous-critère de *testability* vérifie à quel point le logiciel peut être testé pour obtenir des mesures de qualité. Il est important de se préoccuper de cela, car un logiciel facilement testable a plus de chance de ne pas être défectueux. Cela permet aussi d'être plus confiant envers la qualité du logiciel.

- nouveau -

Efficacité de performance

Le critère d'efficacité de performance est le critère qui observe la performance selon la quantité de ressources disponibles. Parmi les sous-critères, les sous-critères de *time behavior* et de *resource utilization* sont importants à considérer.

Le sous-critère du *time behavior* vérifie à quel point les exigences pour le temps de réponse, le temps de traitement et le taux de traitements sont respectées. Ce sous-critère est important, car il ne faut pas que le logiciel prenne trop de temps à déterminer les résultats d'un modèle. De plus, les chercheurs n'ont pas tous des machines derniers cris. Il faut donc que le logiciel fonctionne sur des machines relativement performantes.

Le sous-critère du *resource utilization* vérifie à quel point le logiciel respecte les exigences concernant les ressources qu'il utilise lors de son exécution. Ces exigences concernent le type de ressource ainsi que la quantité utilisée. Ce sous-critère est important, car il est préférable que les ressources de l'ordinateur d'un chercheur ne soient pas toutes prises par le logiciel pour lui permettre de continuer sa recherche en attendant des résultats de ses expériences.

- -

Objectifs, mesures de validation et stratégie de validation

Sous-critère	**Objectifs**	**Mesures de validation**	**Stratégie de validation**
Functional completeness	90-100% des fonctionnalités requises sont présentes	À l'aide des requis, identifier ce qui manque comme fonctionnalités	Tests unitaires
Functional correctness	90-100% des fonctionnalités requises fonctionnent adéquatement	À l'aide des requis, identifier les résultats erronés obtenus après l'exécution du logiciel	Tests unitaires
Maturity	Index de maturité > 0.90	Calcul de l'index de maturité par la formule SMI = Mt - (Fa + Fc + Fd) / Mt	Calcul
Fault tolerance	Le logiciel doit détecter les erreurs 90% du temps	Observer les résultats obtenus lorsqu'il y a des erreurs lors de l'exécution du logiciel	Suite de tests fourni par Weka, tests boite noire.
Recoverability	Le logiciel retourne à l'état stable précédant l'erreur fatale.	Le nombre de fois que le logiciel ne peut pas retourner à un état normal par lui-même	Tests manuels
Analysability	Un score de 4 étoiles selon SIG (système au dessus de la moyenne)	*Unit size*: nombre de lignes dans une méthode 1-15: au moins 57% 16-30: au plus 43% 31-60: au plus 22% 60+: au plus 7%	Analyse du code pour extraire les statistiques, des revues d'inspections pour déterminer le respect des statistiques voulues
		Duplication: redondance: au plus 4.5% non-redondance: au moins 95.5%	
		Unit complexity: nombre de branche dans une méthode (McCabe) 1-5: au moins 75% 6-10: au plus 25% 11-25: au plus10% 25+: au plus 1.5%	

Modifiability	Un score de 4 étoiles selon SIG (système au dessus de la moyenne)	*Duplication*: redondance: au plus 4.5% non-redondance: au moins 95.5%	Analyse du code pour extraire les statistiques, des revues d'inspections pour déterminer le respect des statistiques voulues
		Unit complexity: nombre de branche dans une méthode (McCabe) 1-5: au moins 75% 6-10: au plus 25% 11-25: au plus10% 25+: au plus 1.5%	
		Module coupling: nombre de méthodes appelées provenant d'une classe externe dans une classe 11-20: au plus 20% 21-50: au plus 14% 51+: au plus 6.5%	
Testability	Un score de 4 étoiles selon SIG (système au dessus de la moyenne)	*Unit complexity*: le nombre de branche dans une méthode (McCabe) 1-5: au moins 75% 6-10: au plus 25% 11-25: au plus10% 25+: au plus 1.5%	Analyse du code pour extraire les statistiques, des revues d'inspections pour déterminer le respect des statistiques voulues

- nouveau -

Time behavior	Le logiciel ne doit pas prendre trop de temps pour traiter les données	Le logiciel ne doit pas prendre plus de 30s pour traiter une charge moyenne	Analyse des graphes et des durées produits par JProfiler
Resource utilization	Le logiciel ne doit pas prendre trop de ressources de l'ordinateur	Le logiciel ne doit pas prendre plus que 30% du CPU et de la RAM pour traiter une charge moyenne	Analyser des graphes produit par JProfiler

- -

Collection de tests

Tableau 1 - Tests selon les sous-critères

Sous-critère	**Test**	**Description du test**	**Scope**
Functional completeness	Suite de tests NaiveBayesTest	Vérifier que les fonctionnalités demandées par les utilisateurs/clients sont bien disponibles dans le logiciel. On assume que la suite de test établie au moment de l'élaboration du plan de test initial représente adéquatement les requis.	Algorithme de Bayes Naïf
Functional correctness	testUpdatingEquality()	Vérifier que le modèle produit est le même lorsqu'il est entraîné de façon incrémentale et lorsqu'il est entraîné avec l'ensemble des données dès le début.	Algorithme / classe
Maturity	Calculer la maturité	SMI = Mt - (Fa + Fc + Fd) / Mt SMI: Indice de maturité logicielle Mt: nombre de modules dans la version courante Fa: nombre de modules ajoutés dans la version courante Fc: nombre de modules changés dans la version courante Fd: nombre de modules supprimés comparativement à l'ancienne version	Logiciel
Fault tolerance	Ensemble de tests statistiques StatsTest	L'ensemble des tests présents dans cette classe	Logiciel
Recoverability	Faire planter l'application	Essayer de faire planter l'application pour étudier le comportement post interruption.	Logiciel
Analysability	Extraire facilement un algorithme de Weka	Enlever un module et vérifier que l'application fonctionne encore.	Logiciel
Modifiability			Logiciel
Testability	Plan de tests	Établir un plan de tests, et effectuer les tests tout au long du développement	Logiciel

- nouveau -
-

Time behavior	Création de requêtes sur le logiciel JMeter	Mesurer la performance du logiciel Weka selon 4 scénarios (expliqués plus bas)	Logiciel
Resource utilization	Création de requêtes sur le logiciel JMeter	Mesurer la performance du logiciel Weka selon 4 scénarios (expliqués plus bas)	Logiciel

- -

Résultats de tests

Le sous-critère *functional completeness* ne peut être bien évalué dans le cadre de ce travail puisque les besoins exacts des utilisateurs ne sont pas disponibles. Toutefois, la figure 1 montre bien que l'algorithme de Baye Naïf est bien de type classifier.

Le sous-critère *correctness* a été testé par le biais des tests unitaires. La figure 1 montre que les tests ont été complétés avec succès.

Le sous-critère *maturity* n'a pas pu être calculée dans le cadre de ce laboratoire puisque les données relatives aux variables Mt, Fa, Fc et Fd n'étaient pas disponibles.

Le sous-critère *fault tolerance* n'a pas été testé. En effet, ce genre de test se fait par des tests boîtes noires par des experts qui comprennent toutes les fonctionnalités et les différents algorithme du logiciel Weka.

Le sous-critère *recoverability* n'a pas pu être testé puisque le programme ne fait que se rouvrir lorsqu'il y a interruption d'une simulation. Cette dernière n'est pas sauvegardée et il faut donc recommencer la simulation de nouveau. Le logiciel Weka met à la disposition de ces usagers un fichier d'historiques des actions effectuées dans le logiciel, qui peut permettre de déterminer la cause de la panne.

Les sous-critères *analysability* et *modifiability* ont été testés simultanément par l'ajout d'un nouvel algorithme (MultilayerPerceptron). Par ailleurs, le sous-critère *analysability* aurait dû être testé selon les propriétés suivantes: le volume, la duplication de code et le balancement des composantes. D'autre part, le sous-critère *modifiability* aurait dû être testé selon les propriétés suivantes: la duplication de code, la complexité de l'unité et l'accouplement de module. Toutefois, par manque d'informations sur l'ensemble du logiciel Weka, les propriétés pour ces sous-critères n'ont pas pu être évaluées que cela soit pour les statistiques ou les revues d'inspection.

- nouveau -
-

Le sous-critère *testability* est testé par le biais des nombreux tests unitaires déjà présents. De plus, il faudrait demander à une équipe d'experts d'analyser tout le code pour déterminer sa complexité par manque de temps pour les statistiques et les revues d'inspections.

Les sous-critères *time behavior* et *resource utilization* ont été testés avec JMeter. Les résultats se trouvent plus bas dans la section sur les questions 3 et 4. Pour le *time behavior*, seules les charges augmentée et augmentée exceptionnelle prennent plus que 30s. Cela correspond au critère émis de ne pas dépasser 30s pour des charges moyennes. Pour le *resource utilization*, l'utilisation de la RAM est similaire peu importe la charge de données. Le programme s'alloue 400mb sur 8GB soit à 0,05% de RAM de l'ordinateur pour traiter ses données. Le plan a considérablement surestimé les besoins en RAM de Weka. Quant au CPU, la charge influence le pourcentage consacré à la tâche. Les grosses chargent prennent presque 100% du CPU alors que la petite charge et la moyenne charge ont pris 9.44% et 21.33% respectivement. Cela respecte bien le plan de ne pas dépasser 30% pour une charge moyenne.

- -

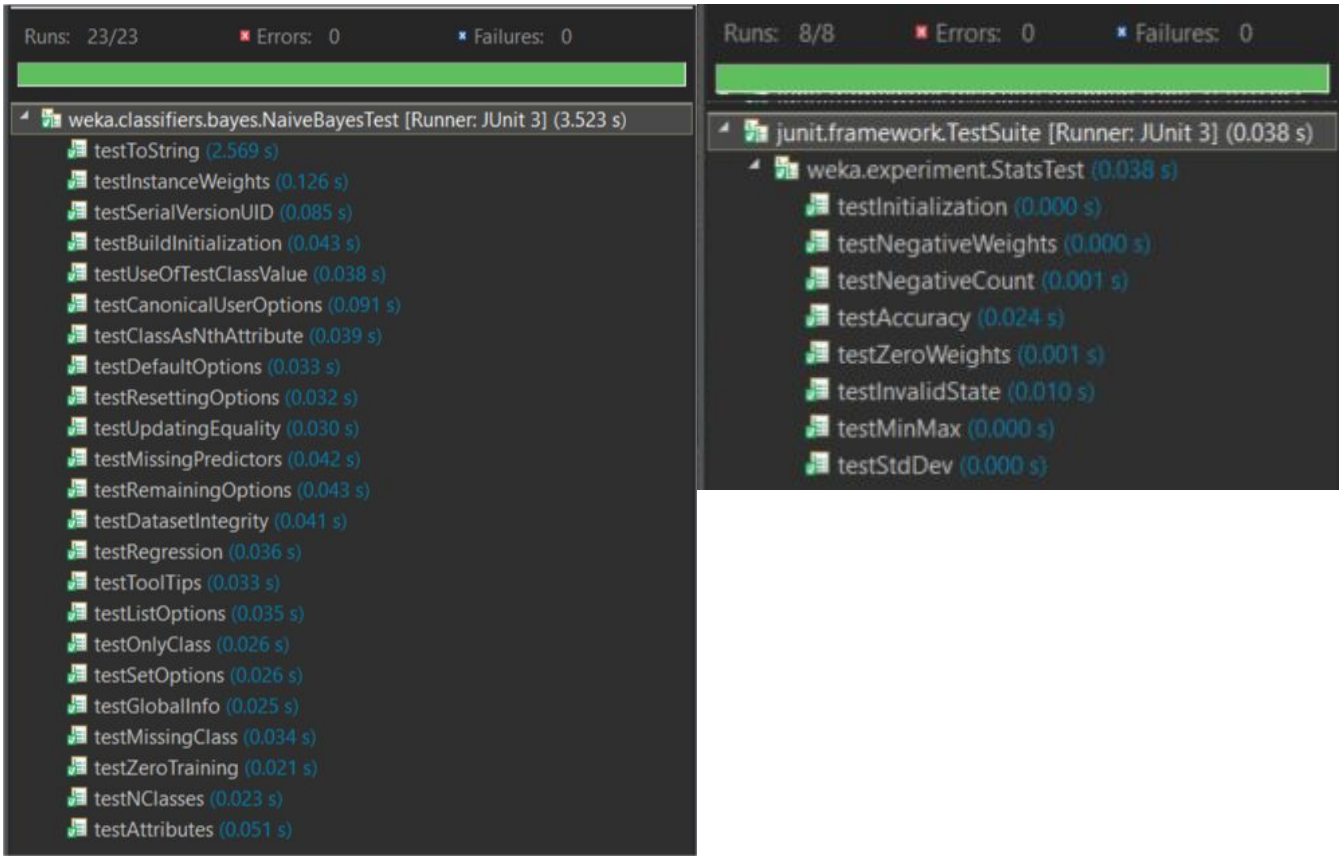


Figure 1 : Résultats des tests unitaires relatifs à l'algorithme de Bayes Naïf

Question 2 : Manuel pour le déploiement de la version REST de Weka sur Docker

1- Ouvrir un terminal
2- Se diriger dans le répertoire : jguwekarest avec la commande : cd ‘path/jguwekarest’
3- Compiler le fichier war (web application archive) avec Maven avec la commande : mvn clean package
4- Créer l’image Docker de REST de Weka avec la commande : docker build -t dockerhubuser/jguweka:OAS3 . (remplacer dockerhubuser par votre nom d’utilisateur de votre compte Docker Hub, ceci est seulement le nom de l’image).
5- Récupérer l’image de MongoDB dans le registre de Docker avec la commande : docker pull mongo
6- (Optionnel) On peut voir les images dans la liste des images avec la commande : docker images
7- Après avoir créé les images Docker nécessaires afin de déployer la version REST de Weka, il faut créer deux conteneurs, l’un pour la base de données avec MongoDB et l’autre pour le logiciel de REST de Weka .
8- Construire et activer un conteneur local de MongoDB avec la commande : docker run --name mongodb -d mongo
9- Finalement, construire et activer un conteneur local de REST de Weka en liant celui-ci avec la base de données MongoDB sous le port 8080 avec la commande : docker run -p 8080:8080 --link mongodb:mongodb dockerhubuser/jguweka:OAS3 (remplacer dockerhubuser par votre nom d’utilisateur de votre compte Docker Hub, ceci est seulement le nom de l’image. Cependant il doit être identique à l’image créé précédemment).
10- (Optionnel) On peut voir les conteneurs dans la liste des conteneurs avec la commande : docker container ls
11- Voilà, le déploiement local de Weka en Docker est effectué. Vous pouvez aller charger la représentation UI dans un navigateur web sur le port 8080 (http://localhost:8080/)

Question 3 : Manuel d'usage de JProfiler

Prérequis: Avoir préalablement suivi les étapes du **Manuel pour le déploiement de la version REST de Weka sur Docker**.

1. Télécharger le logiciel JProfiler en allant sur la page suivante: https://www.ej-technologies.com/download/jprofiler/files. Prendre note de la version utilisée. (Dans ce cas-ci, la version est 11.0 et sera utilisée pour le reste des explications.)
2. Ajouter les lignes ci-dessous dans le fichier Dockerfile pour exposer le port 8849:

```
RUN wget http://download-keycdn.ej-technologies.com/jprofiler/jprofiler_linux_11_0.tar.gz -P /tmp/ &&\
 tar -xzf /tmp/jprofiler_linux_11_0.tar.gz -C /usr/local &&\
 rm /tmp/jprofiler_linux_11_0.tar.gz
ENV JPAGENT_PATH="-agentpath:/usr/local/jprofiler9/bin/linux-x64/libjprofilerti.so=nowait"
EXPOSE 8849
```

3. Recréer l'image Docker :

```
docker build -t dockerhubuser/jguweka:OAS3
```

4. Reconstruire et réactiver le conteneur:

```
docker run -p 8080:8080 -p 8849:8849 --link mongodb:mongodb dockerhubuser/jguweka:OAS3
```

5. Entrer dans le conteneur:

```
docker exec -it [nom du conteneur] bash
```

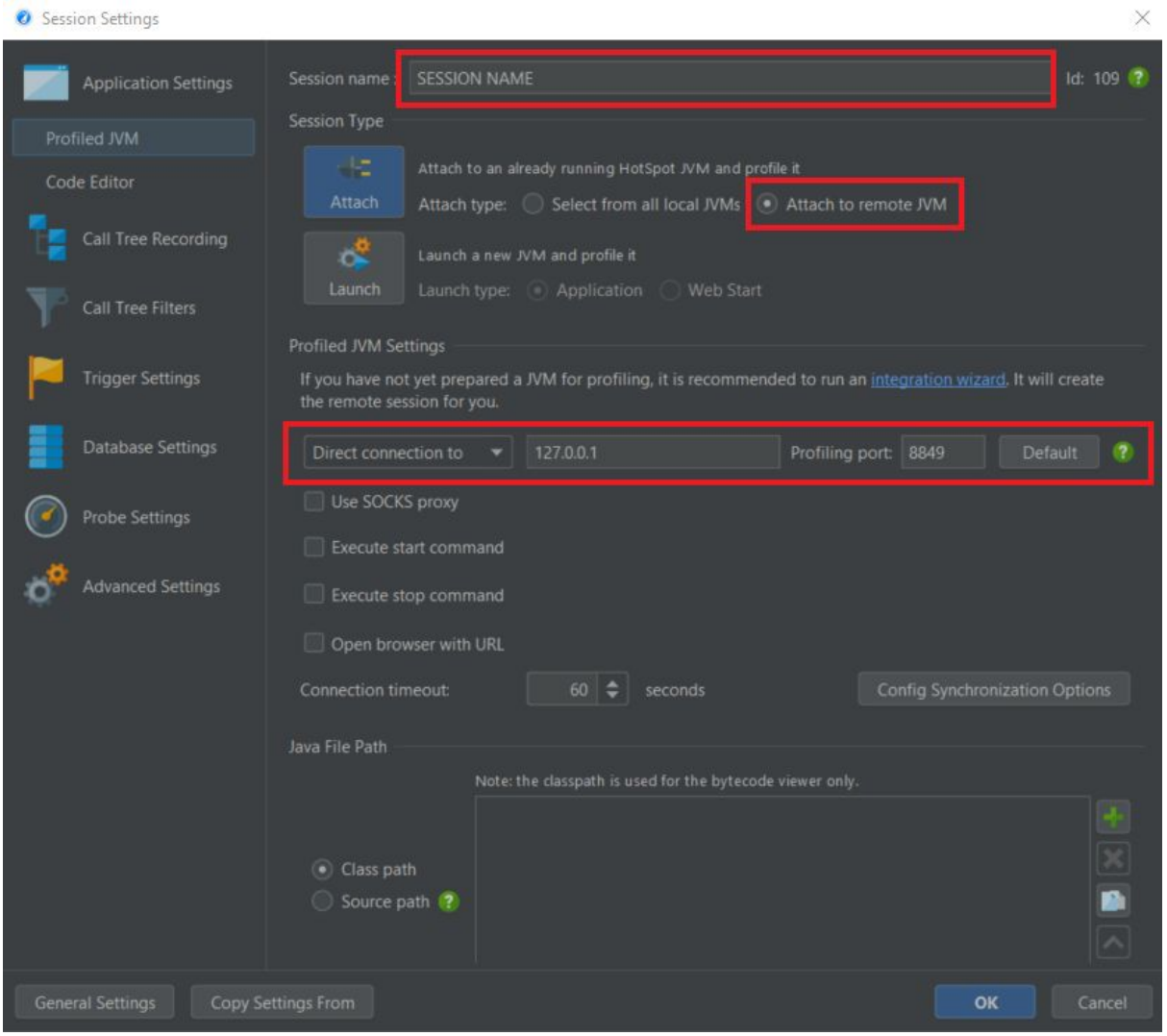
6. Attacher JProfiler au conteneur:

```
cd /usr/local/jprofiler11.0/
bin/jpenable
```

JProfiler va par la suite demander le mode et le port. Entrer "1" et "8849" comme la capture d'écran suivante:

```
tomcat@b73c50530d77:/usr/local/tomcat$ cd /usr/local/jprofiler11.0/
tomcat@b73c50530d77:/usr/local/jprofiler11.0$ bin/jpenable
Connecting to org.apache.catalina.startup.Bootstrap start [1] ...
Please select the profiling mode:
GUI mode (attach with JProfiler GUI) [1, Enter]
Offline mode (use config file to set profiling settings) [2]
1
Please enter a profiling port
[42947]
8849
You can now use the JProfiler GUI to connect on port 8849
tomcat@b73c50530d77:/usr/local/jprofiler11.0$
```

7. Ouvrir l'application JProfiler et créer une nouvelle session.
8. Sélectionner **Attach to remote JVM**. Entrer l'adresse IP, le port 8849 et cliquer OK.



9. Cliquer sur OK lorsque la fenêtre **Session Startup** apparaît. JProfiler est maintenant prêt à être utilisé avec Weka.

Question 4 : Manuel d’usage de JMeter

Prérequis: Une version de Java 8 ou supérieure.

1. Télécharger JMeter (format Binaries) à partir de l’adresse suivante: https://jmeter.apache.org/download_jmeter.cgi
2. Extraire le zip.
3. Ouvrir le folder apache-jmeter-3.0/bin et double-cliquer sur jmeter.bat pour partir l’application sur Windows.

4 scénarios

Charge réduite

Nombre d'usagers: 1

Durée de distribution: 10s

Nombre d'exécution: 2

Nombre de requêtes par usager: 4

Nombre de requêtes total pour la charge: 8

Data: Contact-lenses(24 instances) ⇒ (24 x 1 x 2) = 48 données au total

Charge moyenne

Nombre d'usagers: 5

Durée de distribution: 10s

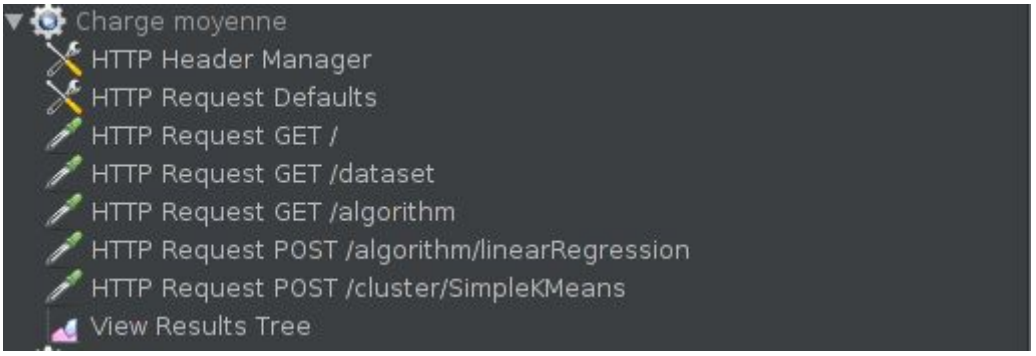
Nombre d'exécution: 5

Nombre de requêtes par usager: 5

Nombre de requêtes total pour la charge: 125

Data: Contact-lenses(24 instances), Iris (150 instances) ⇒ 174 instances

⇒ (174 x 5 x 5) = 4 350 données au total



Charge augmentée

Nombre d'usagers: 10

Durée de distribution: 10s

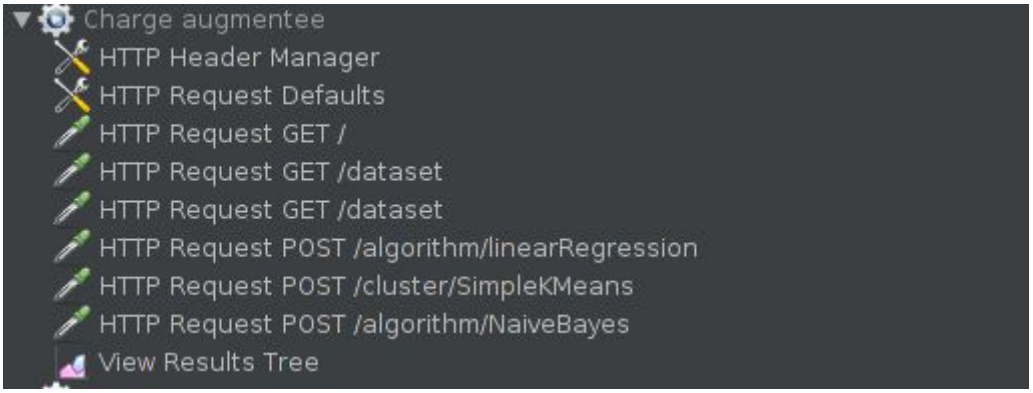
Nombre d'exécution: 10

Nombre de requêtes par usager: 6

Nombre de requêtes total pour la charge: 600

Data: Contact-lenses(24 instances), Iris (150 instances), Glass (214 instances)

⇒ 388 instances ⇒ (388 x 10 x 10) = 38 800 données au total



Charge augmentée exceptionnelle

Nombre d'usagers: 20

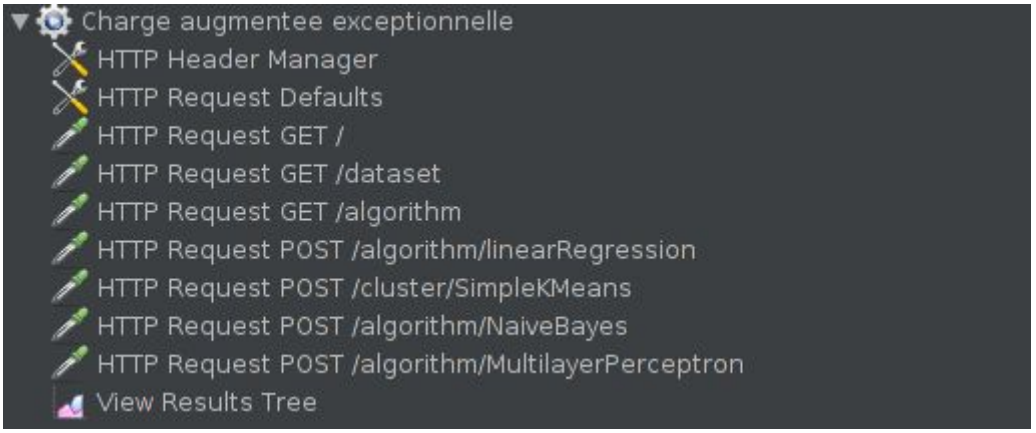
Durée de distribution: 10s

Nombre d'exécution: 10

Nombre de requêtes par usager: 7

Nombre de requêtes total pour la charge: 1400

Data: Contact-lenses(24 instances), Iris (150 instances), Glass (214 instances), Soybean (683 instances) ⇒ 1071 instances ⇒ (1071 x 20 x 10) = 214 200 données au total



Datasets utilisés:

Iris: 150 instances

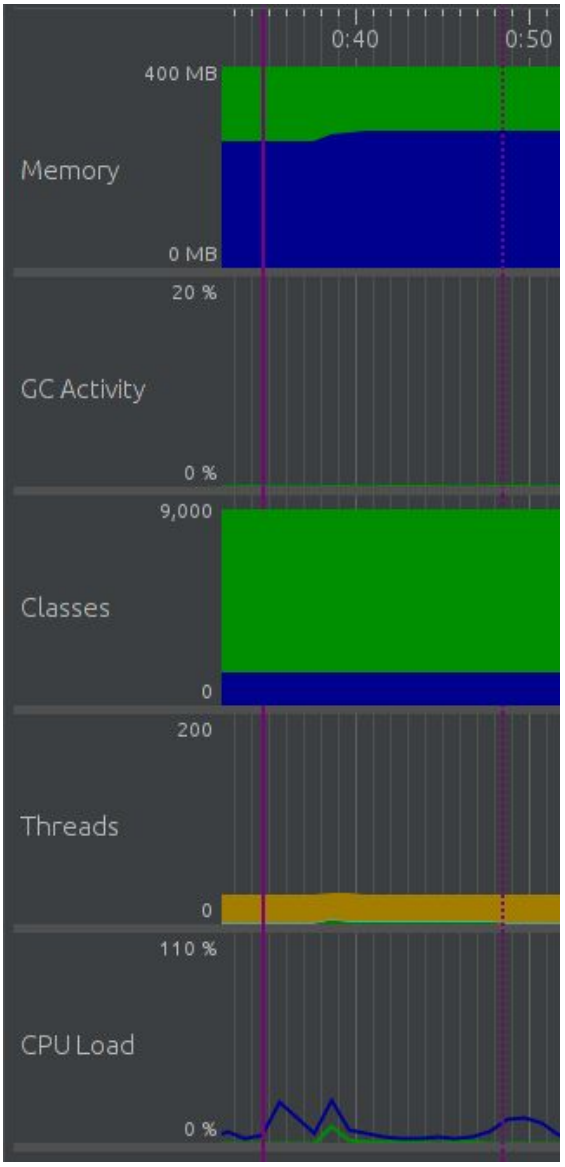
Glass: 214 instances

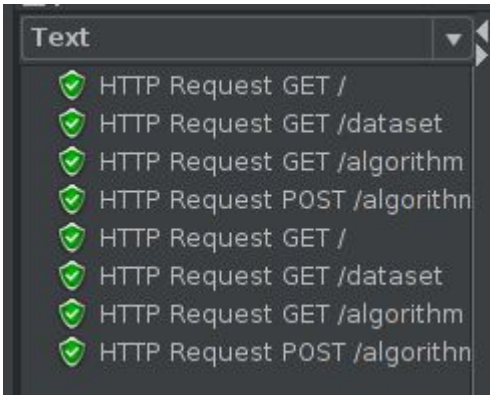
Soybean: 683 instances

Contact-lenses: 24 instances

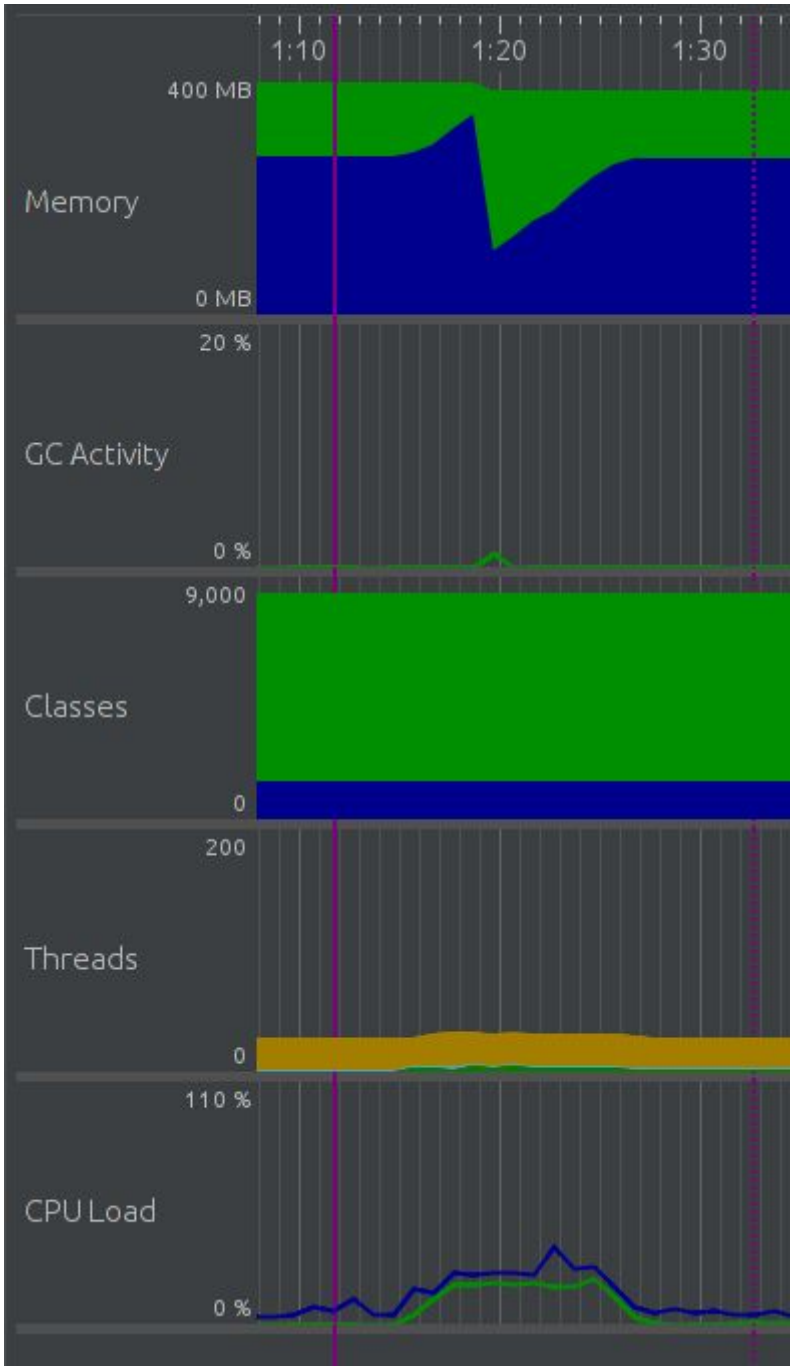
Résultats pour les questions 3 et 4

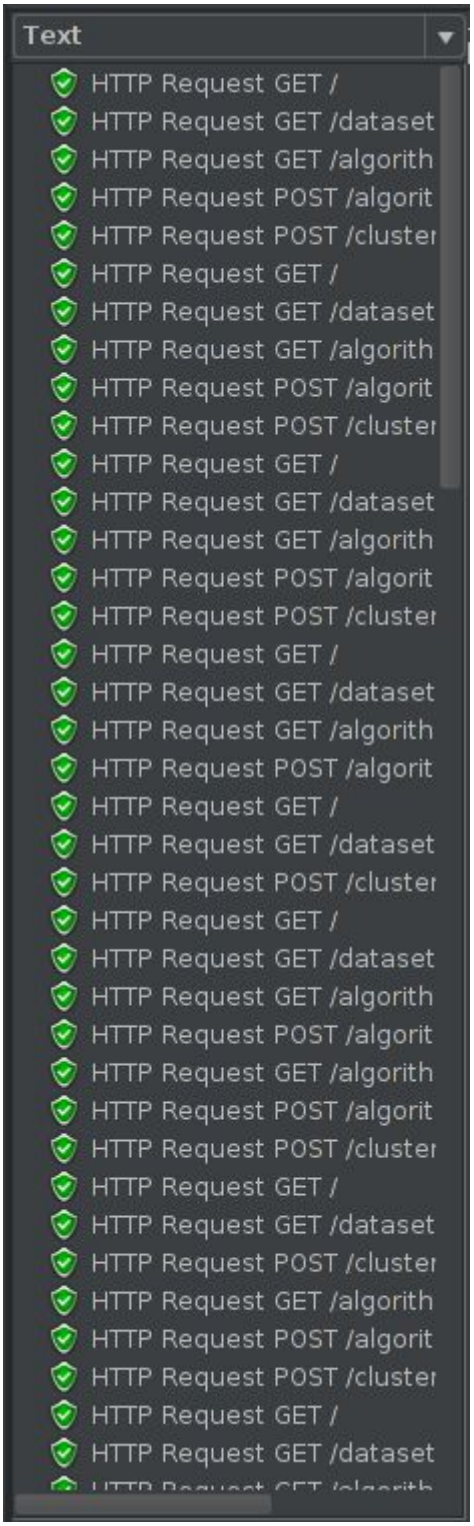
Charge réduite - durée 10s , CPU 9.44%, Ram 400 MB alloué



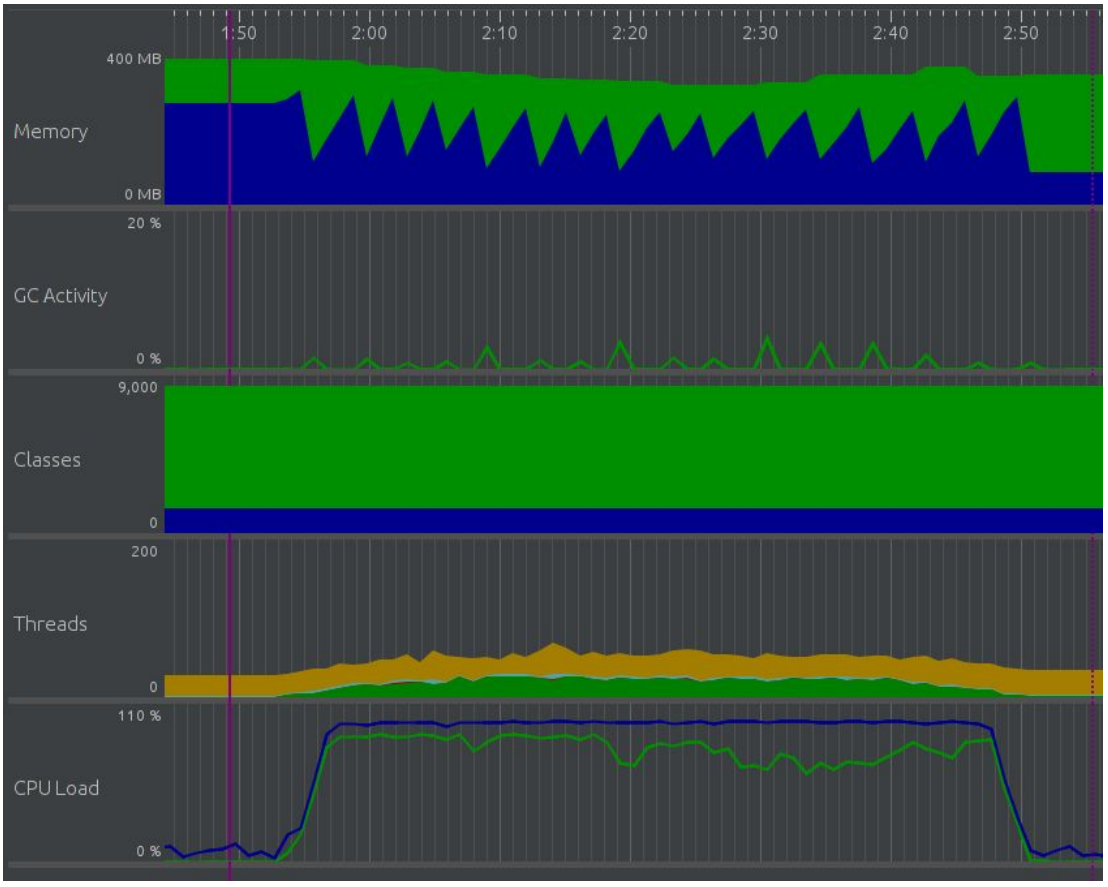


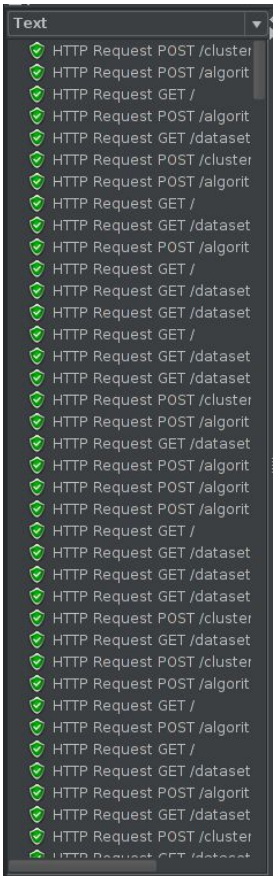
Charge moyenne - durée 15s, CPU 21.33%, RAM 400 MB alloué



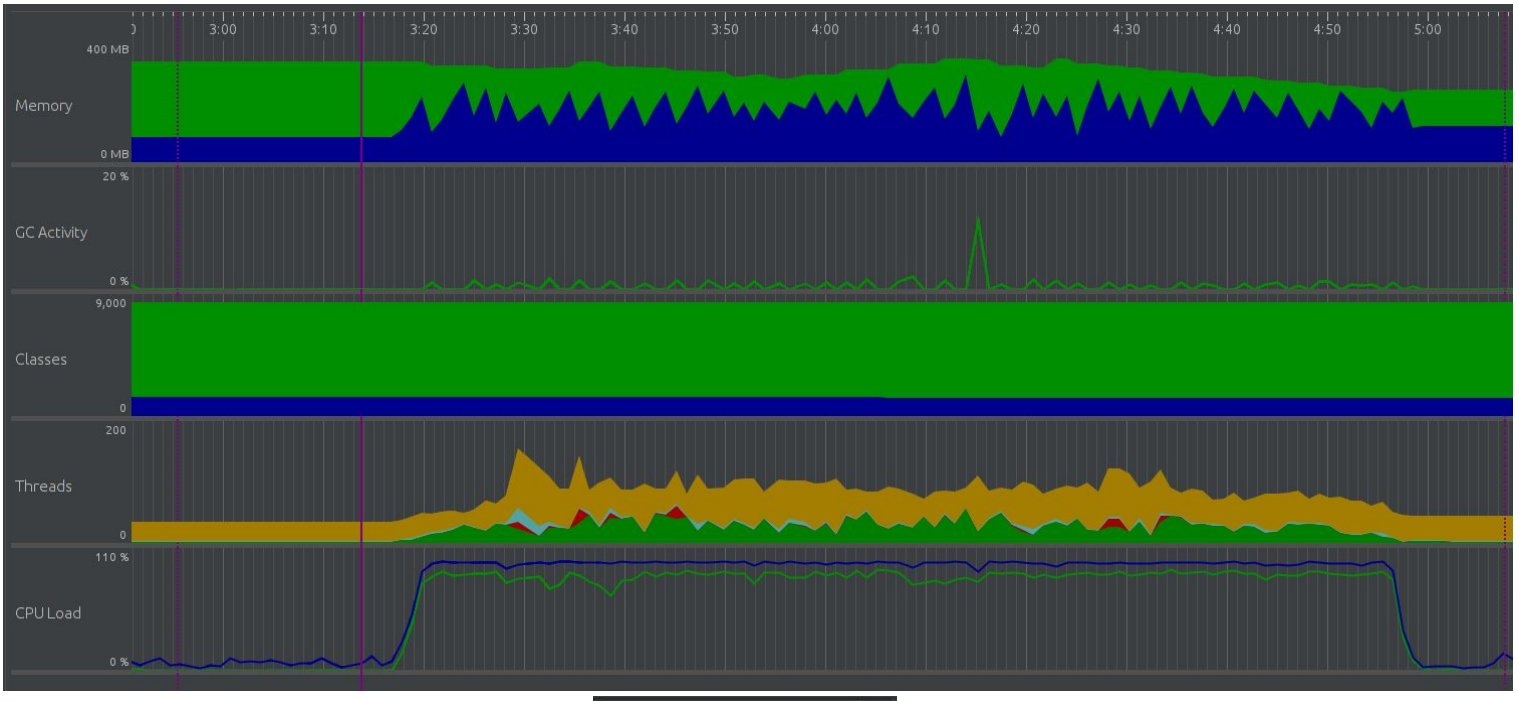


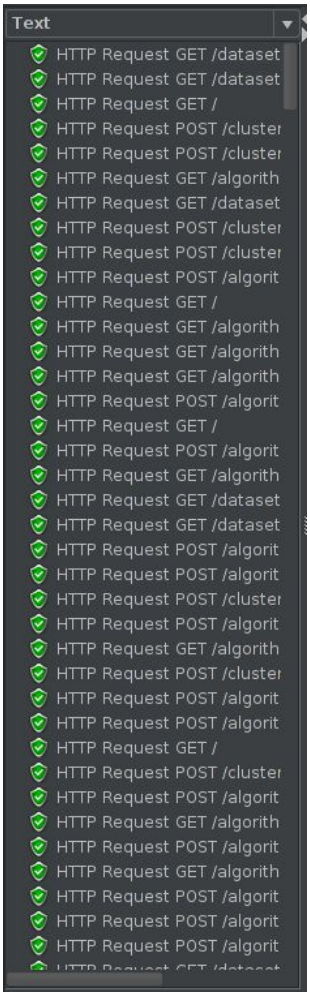
Charge augmentée - durée 1min, CPU 91.11%, RAM 400 MB alloué





Charge augmentée exceptionnelle - durée 1min 44s, CPU 92.59%, RAM 400 MB alloué





Références

[1] Polytechnique Montréal, *Notes de cours*, S.D.

[2] C. Laporte and A. April, *Software Quality Assurance*, IEEE Computer Society, Inc. États-Unis: Wiley, 2018. [En ligne]. Disponible: http://olea.org/recursos/2017-Laporte_April-Software_Quality_Assurance/2017-Laporte,%20April-Software%20Quality%20Assurance,%20First%20Edition-9781119312451.pdf

[3] Orcanos. (2013) Software Maturity Performance. [En ligne]. Disponible: https://www.orcanos.com/compliance/tag/software-maturity/

[4] University of Waikato. (S.D) Weka 3: Data Mining Software in Java. [En ligne]. Disponible : https://www.cs.waikato.ac.nz/ml/weka/