École Polytechnique de Montréal
Département de Génie Informatique et Génie Logiciel

LOG8371
INGÉNIERIE DE LA QUALITÉ EN LOGICIEL
HIVER 2019

Plan d'assurance qualité

Travail pratique no 2
Groupe no 1

Soumis par
Équipe Mike

29 mars 2019

Table des matières

Introduction

Description générale

Weka est une grande collection d'algorithmes d'apprentissage machine très utile pour différentes tâches de minage de donnés. Développé par une équipe de l'université de la Nouvelle-Zélande, ce logiciel à code source ouvert est disponible pour quiconque souhaite en faire usage. Au travers de ses différents algorithmes, Weka met à la disposition de ses utilisateurs différents outils qui permettent de procéder à différentes tâches de traitement. Par exemple, il est possible de préparer les données au traitement, de procéder à leur classification, de faire du regroupement, de mettre en place des règles d'associations pour le minage et puis finalement de visualiser le résultat. Weka est un outil très utile dans le domaine du traitement "Big Data" et de l'apprentissage profond.

Les créateurs de ce logiciel ont comme objectif: de rendre les techniques d'apprentissage machine disponibles à tous; d'appliquer ce logiciel à des problèmes concrets intéressant l'industrie néo-zélandaise; de développer de nouveaux algorithmes d'apprentissage machine et les rendre disponible pour tout; et de contribuer à un cadre théorique pour ce domaine. (Machine Learning at Waikato University, 2019).

Description des fonctionnalités

En ce qui concerne la version abrégée de notre logiciel, les algorithmes qui seront traités sont les suivantes:

Algorithme de classifications: NaiveBayes.java.

L'algorithme de classification Naive Bayes est basé sur le théorème de Bayes. En principe, l'algorithme reçoit des données pour chaque classes et créé un modèle ayant la capacité de créer des prédictions selon les données reçu. Il est important de comprendre que selon l'algorithme, nous supposons que les valeurs d'une classe ne sont pas reliés aux valeurs. Par exemple, si nous voulons calculer la probabilité de vouloir aller à la plage et que nous avons plusieurs classes (météo, distance, température, etc.), l'algorithme va pouvoir calculer la probabilité d'aller à la plage selon ces classes.

En ce qui concerne l'architecture de Weka, afin qu'un algorithme de classification soit intégré au projet, il faut qu'elle implémente la classe *AbstractClassifier*. Cette classe offre une infrastructure qui sera compatible avec les tests déjà implémentés. Le fichier NaivesBayes.java permet de construire un classificateur utilisant la fonction *buildClassifier*(). Elle contient aussi une fonction pour mettre à jour le classificateur et une fonction pour générer la distribution de

probabilité de classe prédite. En principe, cet algorithme doit pouvoir générer un modèle de prédiction en fonction des données reçues.

Algorithme d'associations: Apriori.java

L'algorithme d'association est une méthode de découvrir des relations entre différentes variables dans une grosse base de données et générer de nouvelles règles après l'avoir analysé. Le fichier Apriori.java exécute l'algorithme de type apriori. En autre mots, lorsqu'un item apparaît souvent dans une base de données, nous pouvons l'associer à une certaine règle. Si la fréquence d'apparition est haute, la confiance de cette règle augmente.

Dans notre cas, les différents algorithmes de type association implémentent la classe abstraite *AbstractAssociator.* De plus, en utilisant la fonction *buildAssociations()*, l'algorithme va construire une liste de règle d'association en fonction des données d'entraînement.

Algorithme d'associations: FPGrowth.java

Un autre type d'algorithme d'associations, celui-ci est une méthode plus efficace et évolutive utilisant un arbre de patrons fréquents (FP-Tree). En principe, les noeuds de l'arbre représentent les objets et les branches représentent les associations. Cet algorithme implémente aussi la classe abstraite *AbstractAssociator* et la même fonction que celle mentionné ci-dessus est lancé pour construire une liste de règles d'association.

Algorithme de regroupement: Cobweb.java

Un algorithme de regroupement permet de regrouper des points de données selon certain critères. Les données regroupées ensemble ont en général des propriétés similaires et celles qui sont séparées ont des propriétés très dissimilaire. Celui implémenté dans notre logiciel est basé sur l'algorithme COBWEB, une méthode qui organise les données dans un arbre de classification. Cet arbre peut être utilisé pour prédire la classe de nouveaux objets ou des attributs manquants.

En ce qui concerne notre système, les algorithmes de regroupement doivent tous implémenter la classe *AbstractClustering* et l'utilisateur pourra construire le regroupement des données en utilisant la fonction *buildClusterer()*.

Algorithme de clustering: FarthestFirst.java

L'algorithme implémenté dans ce fichier est une variante de K Means. Il place chaque nouveau centre d'un groupe le plus loin possible des autres centres de groupes existants. Comme mentionné ci-dessus, cet algorithme implémente aussi la classe abstraite *AbstractClustering* et l'utilisateur construit peut construire le regroupement des données en utilisant la fonction *buildClusterer*().

Objectif

L'objectif de ce plan de qualité est de décrire les différents critères que nous allons évaluer ainsi que les sous-critères et métriques que nous allons appliquer. De plus, ce plan contient une description de la stratégie de validation qui devra être utilisée pour assurer le niveau de la qualité du logiciel.

Importance de qualité

Puisque l'utilisation de données est un élément vital d'une entreprise, être capable de faire des décisions en fonction de ces données peut être la différence entre défaire la compétition ou bien perdre la course. Des algorithmes d'apprentissage machine peuvent permettre d'analyser et manipuler les données de manières plus efficaces qu'un être humain. Cette analyse pourra aider les utilisateurs à faire des prédictions et faciliter les décisions à prendre. Ces algorithmes peuvent toucher plusieurs domaines, y compris la santé, les entreprises financières, les infrastructures électriques, etc. Pour cette raison, ce logiciel doit non seulement permettre aux utilisateurs de faire des prédictions de manières précise, mais doit aussi être facilement testable et maintenable lorsque les algorithmes existants doivent être modifiés ou lorsque de nouveaux algorithmes doivent être ajoutés.

Partie prenantes

Ce logiciel inclut plusieurs différentes parties prenantes principales. Premièrement, les principaux utilisateurs de ce logiciel, en ce qui concerne son application, sont les clients: les chercheurs et étudiants universitaires, les entreprises qui appliquent les algorithmes d'apprentissage machines et autres utilisateurs qui pourraient en bénéficier. Deuxièmement, pour ce qui est de la maintenance, développement et qualité du logiciel, les parties prenantes principales sont les développeurs qui implémentent les différents algorithmes, les testeurs qui s'assurent de la qualité du logiciel et les analystes qui s'assurent que le logiciel est bien maintenu et distribuable.

Critères de qualité

Pour ce plan, nous allons seulement traiter les critères suivant: maintenabilité, fiabilité et aptitude fonctionnelle. Le tableau suivant résume les sous-critères et objectifs en lien avec les critères mentionnés.

Critères	Sous-critères	Objectifs
Maintenabilité	Simplicité	La complexité cyclomatique ne doit pas dépasser 25.
	Modularité	L'indice de modularité ne doit pas dépasser 0.4. Le rapport du couplage efférent et le couplage total d'un paquet doit être plus petit que 0.5.
Fiabilité	Récupérabilité	Le temps nécessaire au système pour rétablir son état dans le cas d'une panne ou interruption ne doit pas être plus que 60 secondes.
	Tolérance de fautes et échecs	Le temps moyen entre les échecs rapportés par les utilisateurs doit être plus grand que 5 jours pour une période de 1 mois. Les nouvelles versions du logiciel ne doivent pas causer des anciens problèmes de surgir. Tous les tests unitaires et d'intégration qui passaient dans une version précédente doivent passer dans les nouvelles versions.
Aptitude fonctionnelle	Complétude fonctionnelle	Les fonctionnalitées des algorithmes doivent répondre aux besoins décrits par les catégories dont ils appartiennent à 100%. Ex: Un algorithme de catégorie d'association doit pouvoir générer les règles indépendamment de la méthode.
	Précision fonctionnelle	Le système est capable d'exécuter tous les tests unitaires à un taux de succès de

		90%. Le système doit être capable d'exécuter tous les tests d'intégrations avec un taux minimum de succès de 90%.
Efficacité de performance	Comportement par rapport au temps	Le système doit pouvoir exécuter un algorithme dans un délai qui n'excède pas 3 minutes (pour un fichier de données de 55 Mo et moins).
	Utilisation des ressources	Le système doit en moyenne utiliser 1Go de mémoire vive ou moins pour un fichier d'environs 55 Mo. De plus, il est attendu que le système utilise la majorité de ses capacités pour effectuer ses tâches.
	Capacité du système	Le système doit être capable de traiter des fichiers ayant une grosseur de 55 Mo et moins.

Maintenabilité

Maintenabilité est défini comme étant le degré d'efficacité et d'efficience avec lequel un produit ou un système peut être modifié par les responsables de la maintenance[1]. Selon cette définition et la portée du projet, la simplicité et la modularité sont les sous-critères sélectionnés.

La simplicité d'un logiciel décrit la complexité de comprendre le code source et les algorithmes implémentés. En d'autre mots, si le code source est facile à comprendre et lire, le coût et l'effort de la maintenance devrait diminuer (un nouveau développeur pourra s'intégrer plus facilement à l'équipe). Ce sous-critère sera quantifiable durant les revues formelles du code par les membres sénior de l'équipe de maintenance et en utilisant un outil pour calculer la complexité cyclomatique du code. Avec cette métrique, les évaluateurs pourront évaluer la complexité des fonctions et savoir si le code devra être réorganisé. Plusieurs outils existent pour générer ce rapport par exemple : SonarQube, JaCoCo et le plugin Eclipse Metrics. Il n'y a pas de chiffre magique pour cet objectif, mais selon un des développeurs de Microsoft et une des règles du "Code Analysis", 25 est une bonne valeur pour débuter.

[1] ISO/IEC FDIS 25010, Systems and software engineering - Systems and software Quality Requirements and Evaluation - System and software quality model (2010)

La modularité d'un logiciel décrit l'indépendance des différents modules et classes d'un logiciel. Il est plus efficace et facile de maintenir et modifier un logiciel ayant un niveau de couplage bas entre les classes et modules. La modularité met de l'emphase sur la séparation des fonctionnalités d'un logiciel en modules indépendants et interchangeables (chaque module n'a pas besoin d'un autre pour exécuter sa fonction). Plusieurs langages de programmation supportent ce style de programmation, incluant C++ et Java. Puisque Weka est programmé en Java, nous pouvons l'évaluer. Bien que nous pouvons utiliser la même métrique mentionné ci-dessus, nous recommandons d'utiliser l'indice de modularité défini dans le journal *Revised Modularity Index to Measure Modularity of OSS Projects with Case Study of Freemind.* Cette valeur nous donne un aperçu du niveau de modularité du système et peut aussi indiquer la difficulté d'un tel système. Dans la Fig 9: *Evolution of Modularity Index in Freemind (*Wahju Rahardjo Emanuel, A. Jahja Surjawan, D. International Journal of Computer Applications), nous pouvons observer que l'index de modularité ne dépasse pas 0.4 lors de leur récolte de données. Pour cette raison, nous avons choisis cette valeur comme limite maximale.

Un autre objectif inclut que le rapport entre le couplage efférent et le couplage total d'un paquet doit être le plus petit possible. Plus ce ratio est petit, plus notre paquetage est dit stable et plus un paquetage est modifiable sans avoir un effet sur d'autres paquetages. Pour notre métrique nous avons indiqué qu'une valeur plus petite que 0.5 devrait nous donner un système assez stable tout en nous donnant une bonne marge dans le cas où le système serait plus complexe.

Fiabilité

La fiabilité est "la capacité d'un système ou composante de performer sa fonction sous les conditions spécifiés dans les temps spécifiés"[2]. Pour ce logiciel, les deux sous-critères choisis sont la tolérance d'échecs et la récupérabilité.

En ce qui concerne la tolérance d'échecs, il y a plusieurs métriques que nous pouvons utiliser pour évaluer la fiabilité du système, mais pour notre système, puisqu'il est de type "open source", nous allons utiliser les données récoltées des utilisateurs entre chaque modification (déploiement) ou version. Utilisant ces données, nous pouvons calculer le temps moyen entre les échecs. En autres mots, chaque fois qu'une panne est enregistrée, nous ajoutons ces données à notre banque de données. Utilisant cette banque, nous pouvons construire un graphe de taux d'échecs en fonction du temps. Ceci permettra d'évaluer les activités potentielles qui causent ces pannes et celles qui permettent de réduire ce taux. De plus, puisque les parties prenantes peuvent inclures de grandes entreprises, il est très important que le taux d'échecs soit bas. Une panne peut avoir un impact majeur sur ces entreprises. Dans notre cas, nous avons évalué que 5 jours entre échec devrait être atteignable si nous estimons qu'une période entre deux versions est de 1 mois. De plus, un autre objectif pour ce

[2] ISO/IEC FDIS 25010, Systems and software engineering - Systems and software Quality Requirements and Evaluation - System and software quality model (2010)

sous-critère est que les anciens test qui passaient dans les versions du logiciel précédentes doivent aussi passer. Il ne doit pas y avoir de régression.

La récupérabilité est le temps nécessaire pour que le système rétablisse son état optimale. N'importe quel système doit avoir une façon de récupérer sa fonction après une interruption ou un échec. Comme indiqué ci-dessus, le temps que le logiciel est hors d'usage peut avoir un impact sur les entreprises qui l'utilisent. Dans notre cas, la stratégie que nous employons est d'appliquer des tests de robustesse en utilisant d'énormes quantités de données. De cette manière, nous pouvons évaluer la quantité de données nécessaire pour causer un échec ou une panne et la quantité de temps pour s'y remettre. Ces tests de robustesse se feront avant chaque déploiement d'une nouvelle version du logiciel.

Aptitude fonctionnelle

L'aptitude fonctionnelle est défini comme étant la capacité à un système de fournir les fonctions requisent sous certaines conditions spécifiés[3]. Pour ce logiciel, les deux sous-critères choisis sont la complétude fonctionnelle et la précision fonctionnelle.

La complétude fonctionnelle implique la capacité au système de permettre l'exécution des tâches selon les normes spécifiés. Notre algorithme de clustering doit permettre de faire des regroupements d'objets similaires, l'algorithme d'association doit permettre de générer des règles d'association et l'algorithme de classification doit pouvoir créer un modèle probabiliste qui permet d'estimer la probabilité d'un certain résultat. Chaque catégorie a des interfaces spécifiques qu'il faut utiliser pour implémenter les différents algorithmes. Donc, afin de bien évaluer si notre logiciel réponds à notre sous-critère de complétude fonctionnelle, il faut que chaque algorithme implémente l'interface et soit capable d'exécuter l'algorithme. De plus, les tests déjà implémentés pour une certaine catégorie devront être compatibles avec n'importe quel nouvel algorithme ajouté à cette catégorie. Nous avons fait cette évaluation en supposant la nature des algorithmes d'apprentissage machines et le type de projet que WEKA représente.

Puisque WEKA est un logiciel qui utilise des algorithmes d'apprentissage machine, la précision des données analysées est essentielle pour son utilisation. Cependant, c'est aussi pour cette raison que nous ne pouvons pas avoir une précision de 100%, surtout lorsque nous travaillons avec d'énormes données. Ce que nous pouvons faire dans ce cas est de mettre un seuil de précision pour les résultats des tests. Si le taux de succès des tests unitaires et d'intégration est de plus que 90%, nous pouvons conclure que notre logiciel a atteint nos objectifs. Ceci dépends aussi des données utilisées pour l'apprentissage. De plus, les tests de régressions de doivent pas faillir.

[3] ISO/IEC FDIS 25010, Systems and software engineering - Systems and software Quality Requirements and Evaluation - System and software quality model (2010)

Efficacité de performance

La performance logicielle représente la performance relative de notre système face à la quantité de ressource qu'il utilise. Dans le cas du système Weka nous allons nous concentrer plus précisément sur le comportement du système par rapport au temps, l'utilisation des ressources matérielles et finalement la capacité du système.

L'étude du comportement du système par rapport au temps est très utile afin de s'assurer que le système réponde aux requêtes du client avec un délais acceptable. Comme mentionné dans le tableau présentant les critères de qualité, Weka se doit d'être capable d'exécuter un algorithme et d'obtenir ses résultats en moins de 3 minutes (pour un fichier de données de 55 Mo et moins). Il est primordiale que ce délais ne soit pas dépassé puisque cela pourrait créer de la frustration chez les utilisateurs du système, spécialement si une entreprise, par exemple, utilise les résultats générés par les algorithmes. Le 3 minutes est déjà un temps assez long mais nous trouvons que c'est un bon seuil maximal lorsque nous considérons la nature des algorithmes et leurs complexité. Le temps d'exécution d'un algorithme pourra bien sûr dépendre des ressources nécessaire que le système peut accéder. Les objectifs que nous avons mis en place reflète l'utilisation du système Weka déployé dans un conteneur de type docker.

En complément à cette métrique nous nous concentrons également sur l'utilisation des ressources matérielles par le système Weka. Puisque ce système est une grande collection d'algorithmes d'apprentissage machine principalement utilisé dans le domaine de l'apprentissage profond et du minage de données, il est important que le système Weka Rest soit capable de fonctionner à sa plus grande capacité. Cela signifie que pour bien mener à terme ses tâches, Weka Rest doit avoir accès aux ressources matérielles nécessaire afin de limiter son utilisation de mémoire vive à environs 1Go pour un fichier d'environs 55 Mo. Il va s'en dire que le système Weka ne peut pas fonctionner correctement sur le cloud sans l'utilisation d'un serveur dédié et d'un "load balancer". Il permettra de rapidement déléguer les différentes tâches entre les différents conteneurs. Il serait contre intuitif de limiter les ressources disponibles puisque cela réduirait l'accessibilité au système et par le fait même créerait de la frustration chez les utilisateurs.

En dernier lieu nous avons choisi de nous concentrer sur la capacité générale du système Weka Rest. Cette métrique permet de vérifier les limites maximales du système. Il est préférable de ne pas se rapprocher de ces valeurs afin d'assurer une expérience sans ralentissement du système. Ainsi donc, dans le cas du système Weka Rest, nous jugeons qu'il doit être capable de traiter une demande qui fait usage d'un fichier de 55 Mo ou moins sans problème. Les valeurs maximales de notre système seront déterminées plus loins grâce à l'utilisation de l'outil JProfiler.

Stratégies de validation

Afin d'assurer que le niveau de qualité du logiciel soit conforme aux attentes explicitées par ce plan, les stratégies de validations décrites dans cette section seront appliqués. Le respect des critères seront vérifiés et validés à l'aide de revues personnelles, *walk-through*, inspections et audits.

Revues personnelles

Les revues personnelles sont des revues informelles qui seront effectuées par les développeurs après chaque phase de développement. Durant cette revue, le développeur parcours une liste de vérifications à faire couvrant les parties plus critiques et complexes du système. Ceci permet une vérification préliminaire structurée afin d'attraper des défauts avant de procéder aux prochaines vérifications et phases du projet. La liste de vérifications est conçue pour assurer que chaque sous-critère décrit dans la section précédente est respecté. Cette liste peut être mise à jour par le développeur pour ajouter des vérifications pertinentes à faire sur des parties ajoutant de la complexité au système. Cette méthode de validation seule n'est pas assez pour bien assurer la qualité.

Walk-through

Les *Walk-through* sont des revues informelles qui consistent en une présentation du produit révisé dans le but d'avoir des commentaires d'une perspective utilisateur. Parmis les membres présents pour ces revues il doit y avoir un renforceur de standards, un représentant des utilisateurs, un expert de la maintenance, l'auteur des changements qui fera la présentation et un coordinateur qui s'assure de rendre la revue disponible à tout autre membre du projet pour référence future. La présentation donne un aperçu de l'utilisation du produit révisé, ensuite les membres présents donnent leurs commentaires. Suite au *Walk-through*, un document sommaire est créé par le coordinateur et partagé avec l'équipe, indiquant les problèmes et solutions potentiels ayant été relevés durant la réunion et leur degré d'importance.

Inspections

Les inspection sont des revues formelles dirigées par l'équipe d'assurance qualité se concentrant sur le code et la correction d'erreurs. Les membres participants aux inspections doivent inclure l'auteur du document révisé, un testeur, un concepteur, le développeur responsable pour les changements et un modérateur faisant parti de l'équipe d'assurance. Le modérateur devrait avoir de l'expérience avec des projets similaires. La présentation du code sera conduite par le développeur et une liste de vérifications propre au type de code révisé sera parcourue. Suite à l'inspection, le modérateur fournira un sommaire de l'inspection décrivant les erreurs relevées durant l'inspection et leur degré d'importance. S'il n'y a pas d'erreurs, le produit

est approuvé. S'il y a des erreurs, elles doivent être rectifiées, puis approuvées avec une autre inspection.

Audits

Les audits sont les revues les plus formels et seront internes et conduits par l'équipe d'assurance qualité. Ils serviront à vérifier la conformité des documents audités aux standards et normes établis. Lors de l'audit, les auditeurs suivent un plan d'audit pour vérifier que tous les tests ont étés exécutés avec succès et que tout est conforme dans le projet. Les auditeurs documentent leurs observations qui seront utilisés pour la rédaction du rapport d'audit et pour relever des actions correctives à faire. Suite à l'audit, le rapport final est retourné à l'équipe de développement indiquant les actions correctives à prendre et si les documents audités sont approuvés. Dans le cas où des actions sont nécessaires, il pourra y avoir un suivi après ces actions.

Bibliographie

1) ISO/IEC FDIS 25010, Systems and software engineering - Systems and software Quality Requirements and Evaluation - System and software quality model (2010)

2) Machine Learning at Waikato University, (2018) Machine Learning at Waikato University, Tiré de https://www.cs.waikato.ac.nz/ml/index.html

3) Software Testing Help, (2018), What is Cyclomatic Complexity – Learn with an Example. Tiré de https://www.softwaretestinghelp.com/cyclomatic-complexity/

4) Kaur G, Bahl K., (2014). Software Reliability, Metrics, Reliability Improvement Using Agile Process. Tiré de http://ijiset.com/v1s3/IJISET_V1_I3_24.pdf

5) Wikipedia. (2019). Cyclomatic complexity. Tiré de https://en.wikipedia.org/wiki/Cyclomatic_complexity

6) McKenzie, C. (2018).How to calculate McCabe cyclomatic complexity in Java. Tiré de https://www.theserverside.com/feature/How-to-calculate-McCabe-cyclomatic-complexity-in-Java

7) Wahju Rahardjo Emanuel, A. Jahja Surjawan, D. International Journal of Computer Applications. Volume 59– No.12, (2012). Revised Modularity Index to Measure Modularity of OSS Projects with Case Study of Freemind. Tiré de https://arxiv.org/ftp/arxiv/papers/1309/1309.5688.pdf

8) Wahju Rahardjo Emanuel, A. International Journal of Advanced Computer Science and Applications. Vol. 2, No. 11, (2011). Modularity Index Metrics for Java-Based Open Source Software Projects. Tiré de https://pdfs.semanticscholar.org/4ef5/c89a68f3a58d532dbb24516c44c3f95e9d27.pdf

9) Van der Woude, S. (2018). The effect of modularity on software quality. Tiré de http://scriptiesonline.uba.uva.nl/document/659660

10) Rosenberg, L. Hammer, T. Shaw, J. (1998). Software Metrics and Reliability. Tiré de https://www.unf.edu/~ncoulter/cen6070/handouts/minorreport/Rosenberg.html

11) Mandeep, S. (2017). Types of classification algorithms in Machine Learning. Tiré de https://medium.com/@sifium/machine-learning-types-of-classification-9497bd4f2e14

12) Wikipedia. (2019). Association rule learning, Tiré de https://en.wikipedia.org/wiki/Association_rule_learning

13) SINGULAR ME CORP. Apriori vs FP-Growth for Frequent Item Set Mining. Tiré de https://www.singularities.com/blog/our-blog-1/post/apriori-vs-fp-growth-for-frequent-item-set-mining-11

14) George, S. (2018). The 5 Clustering Algorithms Data Scientists Need to Know. Tiré de

https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

15) Wikipedia. (2019). Cobweb (clustering). Tiré de https://en.wikipedia.org/wiki/Cobweb_(clustering)

16) Naboulsi, Z. (2011). Code Metrics – Cyclomatic Complexity.Tiré de https://blogs.msdn.microsoft.com/zainnab/2011/05/17/code-metrics-cyclomatic-complexity/

17) JDepend Analysis. Tiré de http://www.sat4j.org/r17/jdepend/overview-explanations.html