Projet de Découverte de la Recherche

Détection d'élèves en difficulté sur la PLM

Pierre Boinet
Bertrand Duquesnoy

Année 2014–2015

Projet réalisé pour l'équipe de la PLM du Loria de Nancy

Encadrants : Martin Quinson et Gérald Oster

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Boinet, Pierre

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31315376

Année universitaire : 2014–2015

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Détection d'élèves en difficulté sur la PLM

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le 25 mai 2015

Signature :

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Duquesnoy, Bertrand

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31314998

Année universitaire : 2014–2015

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Détection d'élèves en difficulté sur la PLM

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le 25 mai 2015

Signature :

Projet de Découverte de la Recherche

Détection d'élèves en difficulté sur la PLM

Pierre Boinet
Bertrand Duquesnoy

Année 2014–2015

Projet réalisé pour l'équipe de la PLM du Loria de Nancy

Pierre Boinet
Bertrand Duquesnoy
pierre.boinet@telecomnancy.net
bertrand.duquesnoy@telecomnancy.net

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

Loria
615 Rue du Jardin botanique
54600, Villers-lès-Nancy
03 83 58 17 50

Encadrants : Martin Quinson et Gérald Oster

Remerciements

Nous tenons à remercier Martin Quinson, maître de conférences à TELECOM Nancy et chercheur dans l'équipe Algorille au LORIA ainsi que Gérald Oster, maître de conférences à TELECOM Nancy et chercheur dans l'équipe Score au LORIA pour nous avoir proposé ce sujet et avoir été disponibles pour répondre à nos interrogations. Ces quelques mois nous ont permis de nous faire une idée du métier de chercheur, cette première expérience aura été enrichissante.

Table des matières

Introduction

Programmer's Learning Machine

La Programmer's Learning Machine (PLM) est un logiciel en constante évolution développé par Martin Quinson et Gérald Oster depuis 2008. Ce projet s'est étendu dans la mesure où des ingénieurs rejoignent l'équipe afin de faire progresser l'outil. Matthieu Nicolas fait aujourd'hui partie de l'équipe, pendant deux ans. Ce logiciel est disponible en anglais, français, partiellement en brésilien et italien. Le principe de la PLM est d'initier les élèves de TELECOM Nancy à la programmation sur différents systèmes d'exploitation tel que Linux, Mac et Windows, de façon ludique. En effet, cette plate-forme pédagogique et libre nous plonge dans l'univers des Buggles, petites bêtes qui vont agir en fonction des consignes qu'on lui donne, ou plutôt qu'on lui code.

Ces petites bêtes comprennent différents langages : Java, Python, Scala et C. Les quelques 190 exercices se présentent de la façon suivante : une partie leçon, dans laquelle la problématique est exposée ainsi que les méthodes qui vont nous servir à la résoudre. Ainsi qu'une partie code source dans laquelle on va programmer les instructions au Buggle. Cependant, si les ordres qu'on lui donne ne sont pas les bons, la console nous le dira et on devra rectifier le tir jusqu'à ce qu'on atteigne l'objectif de la leçon.

La PLM dispose de différentes catégories d'exercices qui permettent aux étudiants d'acquérir les notions fondamentales de la programmation. Cela va de la simple boucle for au principe de la récursivité en passant par les algorithmes de tri. Ainsi, la PLM convient tout autant à ceux qui n'ont jamais programmé de leur vie qu'à ceux qui ont déjà quelques bases dans leur bagage de compétences. La figure 1 ci-dessous présente l'interface graphique lors du lancement de la PLM, on peut y voir les différentes catégories d'exercices.

Systèmes d'Apprentrissage intelligents

Ce genre d'outil, que l'on appelle les systèmes d'apprentissage intelligents, se doit d'avoir deux caractéristiques majeures. La première étant que l'environnement de travail soit attrayant afin de pouvoir capter l'attention des élèves voire de les captiver. Pour la deuxième caractéristique, il faut bien sûr que cet outil soit propice à l'apprentissage et qu'il y ait derrière de vraies notions à étudier. Tout cela avec le moins d'interventions humaines possible. C'est ainsi que dans le cadre éducatif, cela permet à un petit nombre de professeurs d'encadrer un grand groupe d'élèves.

Il est de plus en plus fréquent d'utiliser la programmation dans le système scolaire, certaines filières la rendent même obligatoire. D'où l'importance de ces systèmes d'apprentissage intelligents. Le nombre croissant d'étudiants en informatique met d'autant plus en valeur ces systèmes et incitent à les développer. S'il n'y en a pas assez à ce jour c'est parce que l'apprentissage de l'informatique est un segment de l'éducation relativement peu répandu comparé à d'autres (commerce, ingénierie générale,...) malgré que la demande en informaticiens soit croissante.

Une autre explication au manque de systèmes d'apprentissage intelligents est que généralement, les problématiques informatiques sont très vite relativement larges. En effet, il y a un nombre infini de solutions pour parvenir à résoudre ne serait-ce qu'un simple problème.

Limite du sujet

Cependant, la limite de ce concept est la même que lors d'une séance de Travaux Pratiques classique : certains élèves n'osent pas demander de l'aide au professeur et vont ainsi rester bloqués et se décourager. La solution à ce problème est le sujet de notre Projet de Découverte de la Recherche, développer un dispositif qui permettrait de repérer les étudiants en détresse.

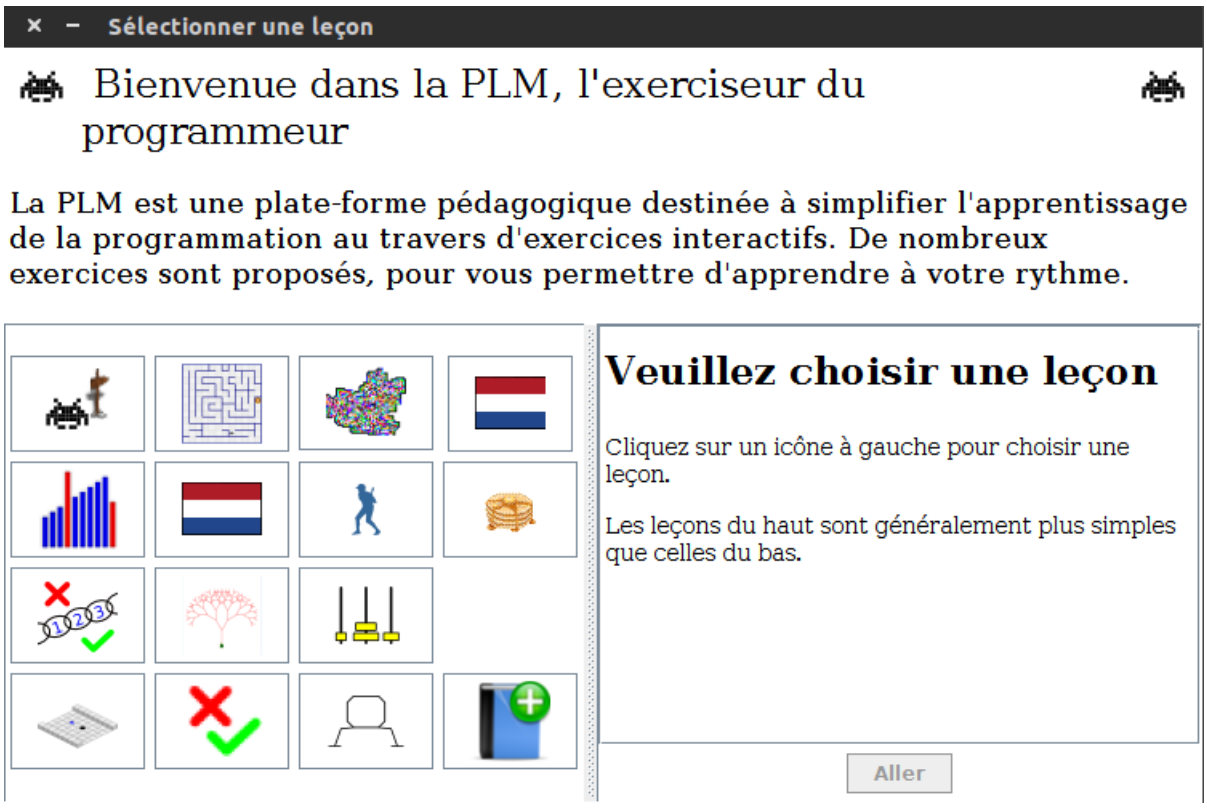

Figure 1 – Lancement de la PLM

1 État de la science dans le domaine

1.1 Motivation des articles lus

Les auteurs des articles que l'on a lus voulaient soit construire un système d'apprentissage intelligent pour un langage particulier (Alice en l'occurrence) soit développer un dispositif qui détecterait automatiquement un élève qui serait bloqué lors de la résolution d'un exercice. Ou encore créer un modèle qui permettrait de prédire le niveau en programmation d'un étudiant en se basant sur la réussite ou non des exercices proposés en début de module. Dans tous les cas ils utilisent une méthode de clustering, qui consiste à étudier les codes sources écrits par les élèves récupérés au cours de leurs évolutions dans les exercices afin des les regrouper.

1.2 Démarches entreprises

Toutes les expériences se sont basées sur un échantillon de minimum 75 élèves qui débutent en programmation. Pendant leurs séances de travail, il fallait différencier les erreurs de compilation et les codes qui compilent mais n'avancent pas vers la solution. Pour se faire ils ont analysé des snapshots (ce sont des captures du code. A chaque compilation de l'étudiant, un snapshot est pris et est commit à un répertoire git. Chaque snapshot est daté et peut être lancé via un simulateur pour enregistrer les erreurs qu'il engendre ou au contraire vérifier que ce code est fonctionnel).

Ensuite, on utilise des algorithmes de clustering (cf. 1.6) pour voir si entre deux snapshots l'élève évolue dans le bon sens ou au contraire s'il se bloque et reste dans cet état. On range finalement ces snapshots dans les clusters après analyse de ces derniers.

Voici un premier aperçu des techniques utilisées dans les articles que l'on a pu lire : utilisation d'algorithmes de comparaison en trois étapes : compte la différence de mots et de dissimilarités dans les deux programmes. S'en suit une décomposition des codes en Arbre Syntaxique Abstrait (ASA) puis une comparaison des deux arbres est effectuée afin de calculer le nombre d'opérations requises pour transformer l'un des ASA en l'autre.

Une autre approche a été de compter le nombre de lignes et caractères ajoutés, supprimés, modifiés entre chaque snapshot. Ensuite ils extraient ces valeurs pour en faire une transformée en z au travers de tous les étudiants pour un exercice donné. Ils utilisent ensuite la méthode de la déformation temporelle dynamique (cette méthode permet de trouver un appariement optimal en fonctions des similarités entre deux séquences temporelles). Puis, ils calculent la distance euclidienne entre les deux snapshots. Enfin, ils regardent si le schéma de programmation de chaque étudiant pour un exercice se rapproche de celui de l'exercice numéro 1, 2, etc.

1.3 Machine learning

Avec les Systèmes d'Apprentissage Intelligents, un programme peut s'adapter à différentes sortes de données sans nécessairement avoir besoin de connaître leur comportement à l'avance. Cet apprentissage automatique se base sur trois points :

1- Observations d'un phénomène

2- Construction d'un modèle de ce phénomène

3- Prévisions et analyse du phénomène grâce au modèle, le tout automatiquement sans aucune intervention humaine.

Les deux catégories sont l'apprentissage supervisé et l'apprentissage non-supervisé.

1.3.1 Apprentissage supervisé

On dispose de différentes classes d'information. On va alors placer une observation X dans une de nos classes, c'est le classement.

Le "scoring" : donner un score d'intérêt à notre information pour dire si elle est plus intéressante qu'une autre ou non.

Exemple : Les spams envoyés à une adresse e-mail. Le système d'apprentissage supervisé va d'abord étudier les mails que l'on a marqué comme spam ainsi que ceux non-marqués afin de se "construire" une idée des mails indésirables. Ensuite, il va regarder les mails définis comme spam afin de créer un schéma d'identification et de mots qui reviennent souvent dans les spams. Parallèlement, des mails acceptés il va déduire des indicateurs de courriers désirables. Finalement, à partir de ces informations, lors de la réception d'un nouvel e-mail, ce dernier sera classé automatiquement.

1.3.2 Apprentissage non-supervisé

Cependant, si les données ne sont pas labellisées, on doit avoir recours à l'apprentissage non-supervisé. L'algorithme se base sur un groupe de données hétérogènes pour trouver des similarités entre elles et les regrouper en plus petits groupes mais cette fois-ci homogènes. En reprenant l'exemple des e-mails, la personne qui examinera ces regroupements (clusters) pourra déterminer si un des groupes est constitué exclusivement de spams afin de le mettre dans la catégorie des courriers indésirables.

Pour notre problématique, on préférera donc un apprentissage non-supervisé. En effet, nous ne savons pas quelle approche un étudiant va avoir lors de son initiation à la programmation, on ne peut donc se baser sur un set prédéfini d'informations sur la personne et en tirer des conclusions.

1.4 Modélisation

Parmi les articles lus, la représentation de la progression des étudiants est modélisée par Markov caché, ou plutôt un automate de Markov à états cachés, modèle statistique. Ce modèle est paramétré par les probabilités qu'a un étudiant de passer d'un état à un autre avec la probabilité qu'un snapshot vienne d'un état particulier. Pour que ce modèle soit cohérent, il faut respecter ces trois conditions :

1- Le nombre d'états doit être fini

2- On connaît la probabilité de chaque transition

3- La probabilité d'émission (voir un snapshot en sachant que l'on est dans un état i)

Pour définir les états de l'automate de Markov, 2000 snapshots ont été analysés puis ont été attribués à des clusters en utilisant l'algorithme de cluster K-medoïdes (cf 1.6). Ensuite, après calcul des probabilités de transitions et d'émissions, le diagramme d'états est construit.

1.5 Méthodes détaillées

Pour obtenir des résultats à partir des techniques expliquées précédemment, nous avons à dispositions différentes méthodes de comparaison. Le but de ces méthodes est de prendre deux fichiers résultants de l'évolution du code de l'utilisateur et de générer un score se basant sur la similarité de ces deux derniers. On construit ensuite une matrice à deux dimensions avec ces scores et c'est ainsi que l'algorithme du Système d'Apprentissage Intelligent peut déterminer ses clusters et y mettre les fichiers correspondant.

Bien que le principe des méthodes soit le même, elles diffèrent par leur fonctionnement. Voici les méthodes de calcul de distance les plus utilisées :

- Bag of words :

On compte les occurrences de chaque mot d'un document, on réitère pour tous les documents produits par un utilisateur. Ensuite on va alors calculer la différence de la valeur des occurrences d'un même mot entre deux fichiers, on procède de même pour tous les mots de ces deux documents (si un mot n'est présent que dans l'un des deux fichiers on garde quand même son nombre d'occurrences). On somme alors toutes ces valeurs pour obtenir la valeur de la distance euclidienne entre ces deux fichiers. On représente généralement ces résultats sous forme d'histogrammes.

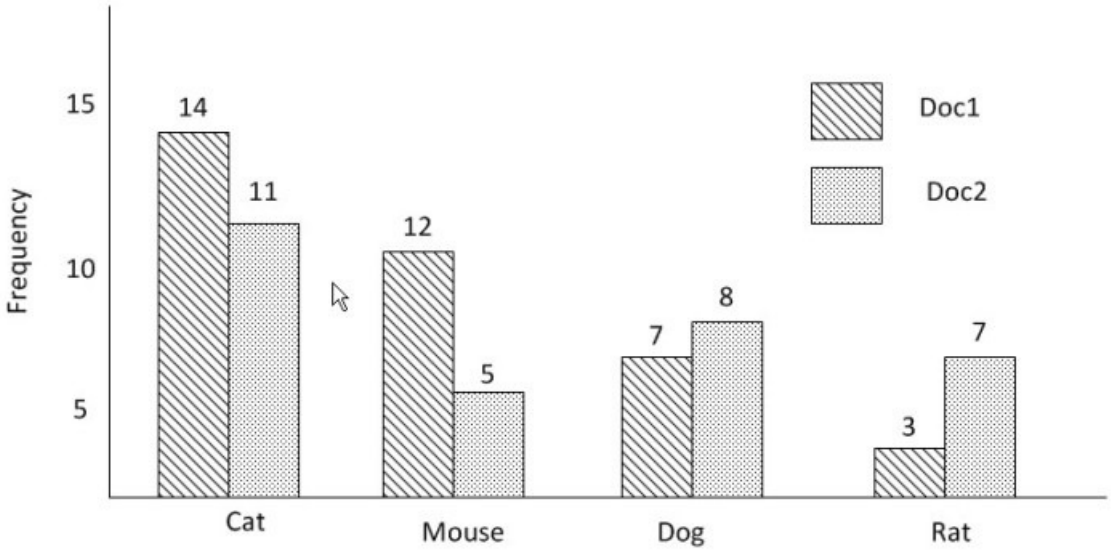

FIGURE 1.1 – Exemple de bag of words
url de l'image

- API call :

Utilise un langage de haut niveau pour analyser les fonctions de plus bas niveau que ce langage appelle. Cette séquence de fonctions est enregistrée et elle est comparée à la séquence d'un autre fichier à l'aide de l'algorithme de Needleman-Wunsch. Cet algorithme utilise le principe de l'alignement de deux séquences de chaînes de caractères afin de calculer le score de la distance.

Exemple :

	A	G	C	T
A	10	-1	-3	-4
G	-1	7	-5	-3
C	-3	-5	9	0
T	-4	-3	0	8

Alors l'alignement :

A	G	A	C	T	A	G	T	T	A	C
C	G	A	C	T	A	G	A	C	G	T

aura le score suivant :

S(A,C) + S(G,G) + S(A,A) + S(C,C) + S(T,T) + S(A,A) + S(G,G) + S(T,A) + S(T,C) + S(A,G) + S(C,T)
= -3 + 7 + 10 + 9 + 8 + 10 + 7 + -4 + 0 + -1 + 0 = 43

Specialized task :

Si le programmeur veut mettre en valeur une certaine partie de la comparaison du code, une fonction spécifique est construite pour y parvenir. Cette comparaison peut être aussi générale que spécifique selon le besoin de l'étude comparative, c'est à lui de le déterminer.

Abstract Syntax Tree Change Severity :

Chaque programme se voit construire un arbre syntaxique abstrait. Entre deux ASA, on calcule le nombre de rotations, insertions et suppressions requises à l'ASA d'un programme pour le transformer en l'ASA de l'autre.

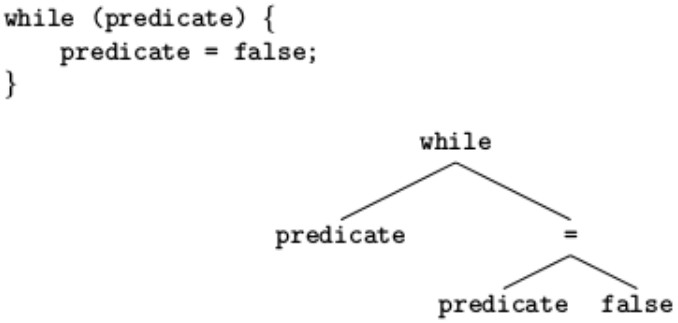

FIGURE 1.2 – Exemple de code avec son ASA
image extraite de l'article *Identifying student stuck states in programming assignment using machine learning*

1.6 Algorithmes de clustering

Quand il s'agit d'algorithmes de clustering, et dans le cas de notre étude, on retrouve les deux suivants : K-medoïdes et propagation d'affinité.

K-medoïdes :

Algorithme de clustering d'apprentissage non-supervisé. Rappelons qu'un médoïde, en statistique, est le représentant le plus central d'une classe. Le but concernant la problématique va être de trouver k clusters dans l'espace de nos données pour ensuite assigner chacune d'entre elles à un cluster. Ensuite, il faut calculer la distance entre chaque centre. Cela peut se faire en utilisant les méthodes décrites précédemment. Cependant cet algorithme a ses limites : nous n'avons pas la garantie qu'il converge et peut se retrouver lui-même bloqué.

L'algorithme fonctionne de la manière suivante :

1- Choisir aléatoirement k données et les définir en tant qu'échantillon initial de centroïdes du cluster

2- Assigner chaque donnée au centroïde dont il est le plus proche

3- Pour chaque centroïde, tenter d'échanger sa place avec chaque non-centroïde et recalculer la distance totale

4- Choisir la configuration la moins coûteuse en terme de distance

5- Répéter les étapes 2 à 4 jusqu'à ce qu'on soit dans la configuration la moins coûteuse

Propagation d'affinité :

Algorithme de clustering d'apprentissage non-supervisé. La propagation d'affinité considère que chaque donnée, représentée par un noeud, peut être considérée comme cluster potentiel. De ce fait il n'est pas nécessaire de choisir un nombre initial de clusters, ce qui dans le cas des K-medoïdes, pouvait nous empêcher de converger vers une solution. Le principe est qu'un noeud A envoie un message à un noeud B contenant des valeurs comparables afin de détecter les similarités entre les noeuds. Ce premier message permet de savoir à quel point le noeud B pourrait être exemplaire pour A. Un second message est échanger entre ces deux points, de B vers A cette fois pour quantifier le fait que B pourrait être l'exemple de A. Cette fois aussi, nous pouvons utiliser les méthodes décrites précédemment afin d'obtenir les matrices de similarité.

L'algorithme fonctionne de la manière suivante :

1- Mise à jour de la fiabilité (premier message)

2- Mise à jour de la possibilité (second message)

3- Répéter les étapes 1 et 2 jusqu'à convergence

1.7 Résultats

La détection d'un élève en difficulté sur la PLM peut se faire de deux façons différentes avec les techniques présentées précédemment. Soit on définit directement des clusters représentant un état de blocage soit on regarde les transitions que prennent les snapshots d'un étudiant entre les états pour déterminer un pattern qui signifierait qu'un étudiant est bloqué.
Il ne faut pas négliger le fait que si un étudiant passe constamment d'un état à l'autre en passant par la même transition, même si ces deux états ne représentent pas un blocage, il faut tout de même considérer l'étudiant comme étant bloqué puisqu'il n'avancera pas vers la solution finale.

Ce que l'on souhaite obtenir ce sont des clusters qui représentent tous un point de progrès ou un état bloqué relatif à la progression d'un étudiant. Si le nombre de clusters est élevé, il y en aura quelques-uns qui seront peu peuplés mais ce n'est pas dérangeant car il n'y aura pas d'effet négatif sur le résultat final. Au contraire, si le nombre de clusters est trop faible, on y perdrait sémantiquement parlant. En effet, un état pourrait se retrouver dans un cluster qui représente un étudiant bloqué alors que ce n'est pas forcément vrai. Il n'y avait pas assez de cluster pour que ce snapshot soit dirigé vers un cluster plus pertinent.

Les résultats après expérimentation diffèrent suivant l'algorithme de clustering choisi et la méthode utilisée. Voici un résumé des résultats qui ont été obtenus (principalement dans l'article Identifying student stuck states in programming assignments using machine learning) :

K-médoïdes :

Aucune des méthodes a permis de trouver un résultat satisfaisant. Le soucis de cet algorithme de clustering est qu'il faut spécifier le nombre de clusters que l'on veut utiliser. Cela cause trop grande disparité des snapshots ou au contraire un rassemblement qui les dénue de leur sens. Les auteurs ont choisi de prendre 16 clusters.

Propagation d'affinité :

Les 4 méthodes ont généré un nombre de cluster différents (entre 12 et 32). Deux méthodes sont sorties du lot : API call et specialized task. API call semble être la meilleure quand on prend en compte tous les critères (nombre de clusters, de snapshots, de transitions, d'états similaires et de transitions non prises). Tandis que specialized task est la méthode qui trouve le mieux l'état de blocage en dépit des autres critères.

2 Matériel et Méthode

2.1 Ressources existantes

Au début du projet, Martin Quinson et Gérald Oster nous ont fourni l'adresse du dépôt github pour que l'on puisse avoir accès à la partie de la PLM qui permet de naviguer dans le code écrit par les utilisateurs de la PLM.

Sur un autre dépôt (PLM-data), on y trouve les commits générés par tout étudiant lorsqu'il démarre la PLM, change d'exercice, fait une erreur de compilation, réussi à passer un exercice, etc.

Format d'un commit :

"kind" :"switched","exo" :"bat.string1.lessons.bat.string1.StringTimes","course" :"",
"switchto" :"sort.pancake.lessons.sort.pancake.BasicPancake","totaltests" :"-1",
"passedtests" :"0","lang" :"Scala"

Le format des commits est JSON, ils contiennent :

- Le type d'évènement

- Le nom de l'exercice courant

- Le nom de la leçon

- Le nom de l'exercice vers lequel l'élève se dirige

- Le nombre total d'essais

- Le nombre de fois que l'exercice a été réussi

- Le langage de programmation utilisé

Parmi toutes les classes que contient le projet PLM-server, nous avons dû au préalable étudier leur fonctionnement afin de cerner celles qu'il fallait que l'on modifie afin de pouvoir mener à bien la mission de notre projet. Les trois classes sur lesquelles nous avons travaillé sont les suivantes :

-GitEvent du package models : c'est la classe qui s'occupe de mettre en forme les commits des étudiants en JSON. On l'utilise pour récupérer des temps correspondant à des évènements qui nous intéressent, comme par exemple l'heure à laquelle un étudiant lance un exercice ou en réussit un.

-Student, du package git.browse : cette classe traite toutes les branches du dépôt. Chaque branche correspond à un étudiant. C'est cette classe qui s'occupe de proposer aux étudiants de donner leur avis sur un exercice concernant son intérêt et sa difficulté. Elle se charge aussi d'incrémenter les évènements d'un étudiant, lorsqu'il exécute son code, qu'il réussit un exercice, etc. C'est dans cette classe que l'on calcule la durée qu'un étudiant passe sur un exercice. On envoie ensuite l'information à la classe Harvester.

-Harvester, du package git.browse : Toutes les informations sont récupérées ici pour les afficher ensuite dans la console. On peut afficher la date, le nombre d'exercices effectués pour chaque langage que propose la PLM, le nombre d'étudiants actifs, le nombre d'exercices réussis, ratés, le nombre de feedback reçus et le nombre d'erreurs de compilation. Il est possible d'afficher ces statistiques de différentes façons. En effet, il existe 4 méthodes, une qui affiche les statistiques quotidiennes, une autre qui affiche celles hebdomadaires, un autre qui nous donne les statistiques mensuelles et enfin la dernière, qui nous fournit les statistiques cumulées depuis la création du dépôt.
C'est dans cette classe que l'on a implémenté l'ajout de nouvelles statistiques.

```
|Date|exos_java|traces_java| exos_python|traces_python| exos_scala|traces_scala|
|----+---------+-----------+------------+-------------+-----------+------------|
|2014.9.2|6|1|0|0|0|0|
|2014.9.3|5|1|0|0|0|0|
|2014.9.4|8|2|0|0|0|0|
|2014.9.8|7|4|0|0|1290|86|
|2014.9.9|0|0|3|1|1615|86|
|2014.9.10|0|0|1|1|163|16|
|2014.9.11|3|2|0|0|999|63|
|2014.9.12|67|4|2|2|1149|67|
|2014.9.13|20|3|1|1|132|14|
|2014.9.14|7|1|0|0|170|12|
|2014.9.15|9|2|0|0|1112|73|
|2014.9.16|16|2|0|0|768|60|
|2014.9.17|36|3|0|0|274|40|
|2014.9.18|1|1|22|2|20|2|
|2014.9.19|1|1|18|2|10|2|
|2014.9.20|0|0|12|1|61|6|
|2014.9.21|0|0|6|2|36|6|
|2014.9.22|3|2|11|2|420|50|
|2014.9.23|1|1|0|0|218|31|
|2014.9.24|0|0|12|2|39|4|
|2014.9.25|0|0|1|1|220|30|
There is 864 non-empty students (+ 53 beta users), 16251 passed exos (of which 652 have a feedback) and 200793 valid commits!
Failed exos: 69622; compil error:39941
```

FIGURE 2.1 – Affichage des statistiques quotidiennes

2.2 Objectifs et méthodes

Notre objectif était de faire des statistiques sur plusieurs métriques afin de pouvoir repérer les étudiants en difficulté en comparant leurs métriques à celles des autres.

Nous avons d'abord commencé par récupérer ces différentes valeurs par étudiant et par exercice à travers la classe Harvester. Pour chaque étudiant était déjà présent le nombre d'essais, d'erreurs de compilation, d'échecs (lorsque l'objectif de l'exercice n'est pas atteint malgré que le code compile bien), et enfin le nombre de lignes du code en cas de succès. Pour la mesure du temps passé sur un exercice nous avons utilisé la différence entre l'heure des commits signalant l'arrivée d'un élève sur un exercice et ceux signalant le départ de celui-ci. Nous avons choisi de ne plus incrémenter le temps passé sur un exercice une fois que celui-ci à été réussi.

Pour l'étude des statistiques, nous avons choisi de nous concentrer sur les données provenant d'élèves ayant réussi l'exercice, afin d'avoir des valeurs plus stables à partir desquelles nous allons pouvoir déterminer des seuils nous aidant à considérer un élève comme étant en difficulté. De même nous nous concentrons sur les exercices ayant un minimum de données (par exemple au moins 100 élèves l'ayant essayé). Nous avons commencé à calculer les statistiques de manière incrémentielle en java, avant de décider qu'il était plus simple d'exporter les données et de les traiter dans un langage plus approprié pour les statistiques tel que R. Nous avons donc exporté toutes les métriques concernant les étudiants dans un fichier csv. Une fois les données importées sous R il est facile d'obtenir la moyenne, le maximum, le minimum, la médiane et les premier et troisième quartiles avec la fonction summary(). Le langage R permet aussi d'exporter facilement des diagrammes de type boîte à moustache permettant facilement de visualiser les données.

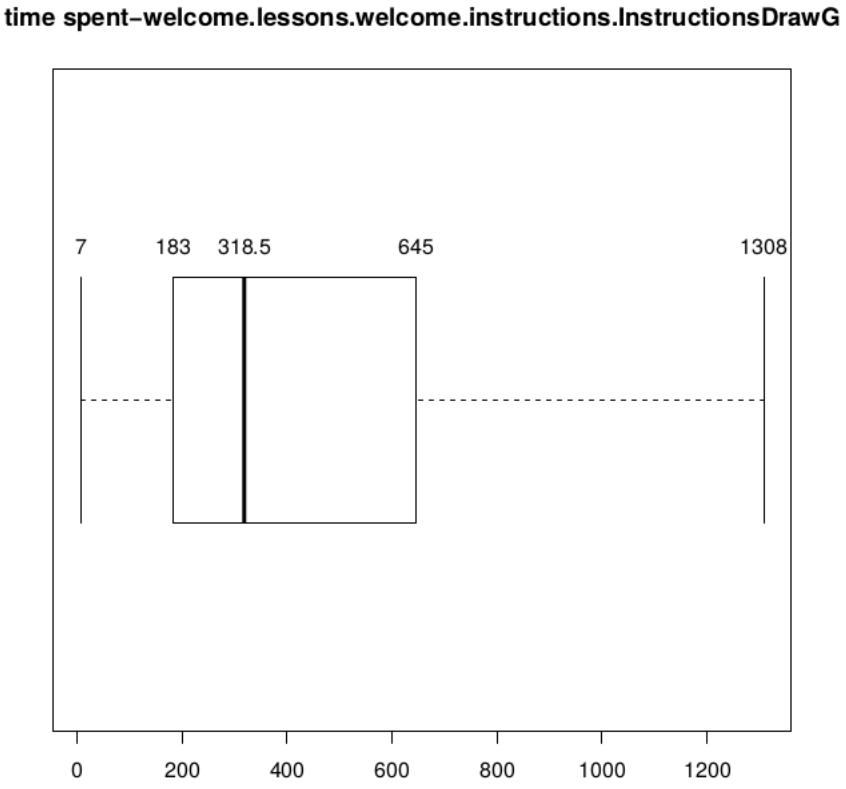

Figure 2.2 – Statistiques de temps en seconde sur un exercice

Ces observations nous permettent de tester différents seuils entre autre en se basant sur les quartiles de différentes métriques afin d'extraire des étudiants potentiellement en difficulté en implémentant dans la classe Harvester une fonction comparant leurs données à différents seuils.

Il est important de se baser sur plusieurs métriques afin de ne pas négliger certains facteurs qui ont leur importance. En effet, un exercice plutôt axé sur la réflexion mais qui peut se résoudre assez vite ne devrait pas mettre un élève en état bloqué parce qu'il met trop de temps à y arriver. Sa réflexion peut se construire un peu plus lentement que les autres mais malgré ses erreurs de compilation, il avance tout de même vers la solution.

3 Résultats et Interprétations

En faisant varier les valeurs des seuils et en essayant de recouper les différentes métriques on a pu extraire différents jeux d'utilisateurs ou plutôt de sessions, cependant il est difficile de vérifier ou de confirmer ces résultats pour plusieurs raisons.

Premièrement par soucis d'anonymat il nous est impossible de faire concorder une session avec une personne réelle, il est donc difficile de calibrer des seuils sans réellement pouvoir les comparer à des résultats d'examen par exemple ou autres valeurs chiffrées permettant de les vérifier ou ajuster.

De plus il existe un nombre conséquent de sessions avec seulement quelques exercices (moins de 5-10) laissant comprendre que certains utilisateurs se retrouvent sur une nouvelle session à chaque utilisation de la PLM. Or il est difficile de considérer un étudiant comme étant en difficulté en se basant sur un ou deux exercices, il est nécessaire que les mesures soient pondérées sur un nombre minimal d'exercices afin d'éviter qu'elles soient biaisées par du bruit.

Ainsi plusieurs types de mesures peuvent venir perturber cette détection, ils peuvent provenir du comportement de l'utilisateur, par exemple il est possible qu'un utilisateur laisse ouvert la PLM sur son ordinateur pendant plusieurs heures créant ainsi des mesures extravagantes. Mais ces parasites peuvent aussi venir du système, en effet il arrive que deux commits proches dans le temps se retrouvent inversés perturbant l'algorithme calculant le temps passé sur un exercice et empêchant d'en calculer une partie voire sa totalité. Par exemple nous avons mesuré que sur 11479 exercices réussis 334 n'ont pas de données concernant le temps passé par l'étudiant sur l'exercice.

Enfin, pour lutter contre ces problèmes il pourrait être intéressant de modifier la PLM pour par exemple détecter lorsque l'étudiant est actif ou si la fenêtre a perdu le focus depuis un temps donné. Mais le problème principal reste la question de l'identification de l'étudiant, essentielle à l'objectif du projet, car une fois une session identifiée comme étant en difficulté le but de la démarche est de pouvoir aider l'étudiant.

Conclusion

Lorsque nous avons commencé le projet, nous ne savions pas en quoi consistait le métier de chercheur. Messieurs Quinson et Oster nous ont permis de bien appréhender le déroulement de ce projet. Après une période à effectuer un travail bibliographique afin de correctement cerner le sujet, nous avons pu nous plonger dans le code de la PLM avec des objectifs clairs en tête.

Ce sujet était notre premier choix, nous trouvions intéressant le fait de travailler sur la PLM dans la mesure où c'est un outil qui nous a aidé à nous former en première année à TELECOM Nancy, lorsque la programmation était encore un terrain inconnu. De ce fait, nous avons pleinement conscience de son utilité.

Cette année, nous sommes deux binômes à travailler sur la PLM, nous sur la détection d'élèves en difficulté et le deuxième binôme sur la remédiation automatisée de masse. Ceci dénote une réelle envie de progression sur un projet qui a débuté en 2008 et c'était d'autant plus motivant pour nous.

Nous avons commencé à explorer une méthode menant vers une solution en déterminant de nouvelles métriques sur les exercices de la PLM. Cependant, pour avoir des résultats plus précis et fiables, il faudrait se diriger vers les méthodes de type clustering qui ont été expliquées le long de ce rapport.

Bibliographie

- Juha Helminen, Petri Ihantola, Ville Karavita, Lauri Maalmi (2012). *How do students solve parsons programming problems ? An analysis of interaction traces.* http://dl.acm.org/citation.cfm?id=2361300. Department of Computer Science and Engineering, Aalto University, Finland.

- Johan Lindell (2014). *Identifying student stuck states in programming assignments using machine learning.* http://www.diva-portal.org/smash/get/diva2:693744/FULLTEXT01.pdf. Linköpings universitet, SE-581 83 Linköping, Sweden.

- Chris Piech, Mehran Sahami, Daphne Koller, Stephen Cooper, Paulo Blikstein (2010). *Modeling How Students Learn to Program.* http://web.stanford.edu/~cpiech/bio/papers/modelingHowStudentsLearn.pdf. Computer Science Department, School of Education, Stanford University, Stanford, CA. 94305.

- Marcelo Worsley, Paulo Blikstein (2013). *Programming Patways : A Technique for Analyzing Novice Programmers' Learning Trajectories.* http://link.springer.com/chapter/10.1007/978-3-642-39112-5_127. Stanford University, Graduate School of Education, Stanford, CA, USA.

- Siba Haidar. *Séries chronologiques et méthodes de comparaison.* http://www.irit.fr/~Philippe.Joly/Homepage_files/Theses/Siba_Haidar.pdf.

- Fabrice Rossi (2009). *Apprentissage supervisé.* http://apiacoa.org/publications/teaching/data-mining/supervised.pdf. TELECOM ParisTech, Paris, France.