Apprentissage de représentations pour l'enseignement de la programmation : une approche centrée enseignant

Guillaume Cleuziou, Frédéric Flouvat, Matthieu Exbrayat, Julien Robert, Romuald Thion

Apprentissage de représentations pour l'enseignement de la programmation : une approche centrée enseignant

Guillaume Cleuziou[1,2,3], Frédéric Flouvat[1], Matthieu Exbrayat[2], Julien Robert[3] et Romuald Thion[1,4]

[1] Université de la Nouvelle-Calédonie, ISEA EA 7484, Nouméa
prenom.nom@unc.nc
[2] Université d'Orléans, INSA Centre Val de Loire, LIFO EA 4022, Orléans
[3] Université d'Orléans, IUT Département Informatique, Orléans
prenom.nom@univ-orleans.fr
[4] Université Lyon 1, LIRIS CNRS, Villeurbanne
prenom.nom@univ-lyon1.fr

Résumé. L'utilisation de plates-formes de validation de programmes est de plus en plus fréquente pour l'apprentissage des bases de l'informatique. Si l'automatisation des tests de validation libère du temps pour l'enseignant et ludifie le parcours de l'apprenant, on observe également certains biais induits par ces dispositifs pédagogiques. Des recherches en Intelligence Artificielle (IA) visent à pallier certains de ces biais. Pour cela, elles se heurtent au problème de l'analyse informatique de ces programmes par les algorithmes d'IA. Cet article traite d'une expérience pratique d'usages pédagogiques d'une plate-forme d'entraînement à la programmation. Nous présentons une nouvelle méthode pour construire des représentations vectorielles des programmes. Dans une approche centrée 'enseignant', nous proposons enfin d'exploiter ces nouveaux espaces de représentation pour détecter automatiquement des programmes validés par la plate-forme mais devant être signalés à l'enseignant.

Mots-clés : Enseignement de la programmation, Apprentissage de représentations, Réseaux de Neurones, SVM.

Abstract. The use of program validation platforms is becoming more and more common for learning the basics of computer science. While the automation of validation tests frees up time for the teacher and facilitates the learner's path, there are also certain biases induced by these pedagogical tools. Numerous works in Artificial Intelligence (AI) aim to overcome some of these biases. For that, they come up against the problem of the analysis of these programs by AI algorithms. In this paper, we rely on a practical experience of pedagogical uses of a programming training platform. We present a new method to construct vectorial representations of programs. In a teacher-centered approach, we finally propose to use these new representation spaces to automatically detect programs validated by the platform but which must be signaled to the teacher.

Keywords: Coding teaching, Representation Learning, Neural Networks, Support Vector Machine (SVM).

1 Introduction

L'utilisation de plates-formes d'apprentissage s'est considérablement développée ces dernières années au sein des formations en informatique, en particulier pour l'enseignement des fondamenaux de l'algorithmique et de la programmation [5, 2, 3]. Celles-ci prennent le plus souvent la forme d'une interface Web sur laquelle un apprenant soumet un programme comme solution à un exercice conçu par son enseignant ; il reçoit, en réponse, le résultat d'une évaluation/validation automatique de son programme. Cet article s'intéresse à cette pratique pédagogique dans sa dimension formative (et non évaluative), autrement dit dans ce qu'elle offre comme leviers pour améliorer l'acquisition des compétences pour l'apprenant.

Dans ce contexte, les intérêts de recourir à ces plates-formes sont multiples et principalement d'ordres quantitatif et qualitatif : l'automatisation d'une partie de l'accompagnement pédagogique peut permettre le pilotage d'un groupe plus large d'apprenants ; déléguer à la plate-forme la validation des programmes permet d'améliorer qualitativement l'accompagnement en consacrant plus de temps à l'analyse/remédiation sur des aspects plus fondamentaux. Du point de vue de l'apprenant, ce dispositif pédagogique apporte une dimension ludique et une autonomie susceptibles de favoriser son implication et sa progression.

Les plates-formes d'entraînement à la programmation présentent cependant des biais importants qui constituent de véritables verrous pédagogiques, opposables à cette modalité d'apprentissage [10]. Le risque majeur pour l'enseignant est de manquer de visibilité et d'exhaustivité dans son appréhension et sa maîtrise du groupe. On observe également un risque pour l'apprenant de s'enfermer progressivement dans une pratique de type essai-erreur improductive et susceptible d'induire une forme de découragement voire de désapprentissage.

Pour y remédier, les recherches actuelles s'orientent vers le dépassement de la simple tâche de validation et tentent d'introduire un réel accompagnement pédagogique, notamment au moyen de *feedbacks* plus nombreux, plus précis et générés automatiquement [11]. Cette ambition s'avère cependant assez limitée en pratique. Ces recherches s'appuient essentiellement sur des techniques d'Intelligences Artificielles (IA) qui se heurtent rapidement au problème de l'analyse des programmes informatiques par les algorithmes d'IA, ces derniers requérant généralement des données (représentations) vectorielles. Au-delà de ce verrou technique, ces recherches reposent sur l'hypothèse forte et risquée que l'expertise pédagogique d'un enseignant spécialisé en informatique peut être (au moins en partie) automatisée et ainsi mimée par une IA. Nous jugeons cette hypothèse 'risquée' pédagogiquement puisque la moindre erreur, par exemple dans un *feedback* généré, impactera directement l'apprenant.

Dans ce contexte, les contributions présentées dans cet article sont les suivantes : à partir de l'observation des usages d'apprenants et d'enseignants sur une plate-forme d'apprentissage, nous proposons d'exploiter des techniques récentes d'IA avec un point de vue centré 'enseignant'. L'objectif devient ainsi d'offrir à l'enseignant une assistance au pilotage et au développement de sa pratique pédagogique à travers la plate-forme. Ce positionnement présente trois avantages majeurs :

1. répondre conjointement aux deux verrous pédagogiques mentionnés : redonner de la visibilité à l'enseignant et lui permettre ainsi d'intervenir efficacement auprès des étudiants en situation de blocage,
2. éliminer les risques pédagogiques : l'enseignant conservant la maîtrise des interventions auprès des apprenants,
3. s'affranchir de l'hypothèse selon laquelle il existerait *une unique* pratique d'accompagnement pédagogique qui serait de surcroît automatisable.

Cette étude s'appuie sur une plate-forme d'entraînement à la programmation dont nous présentons le principe général de fonctionnement en section 2. Nous discutons des principaux biais pédagogiques observés (Section 3). Nous réalisons ensuite une preuve de concept sur la possibilité d'introduire de l'IA dans une approche centrée enseignant. Cette contribution repose sur une nouvelle méthode d'apprentissage machine (*machine learning*) [4] adaptée à l'analyse de programmes informatiques (section 4). Celle-ci est ensuite utilisée pour détecter automatiquement des programmes validés par la plate-forme mais sur lesquels un enseignant pourrait juger intéressant d'intervenir. Cette expérimentation encourageante, menée sur plusieurs milliers de programmes Python d'étudiants débutants en programmation, est présentée en Section 5.

2 Plate-forme d'entraînement à la programmation

Cette étude repose sur l'usage pédagogique d'une plate-forme d'entraînement à la programmation (ou exerciseur) développée à l'IUT d'Orléans. Elle prend la forme d'une interface Web sur laquelle les étudiants saisissent et évaluent leurs solutions aux exercices proposés. Bien que la plate-forme offre un large spectre de langages (Python, Java, SQL, etc.), nous limitons notre étude à l'apprentissage de la programmation et illustrons ce travail avec le langage Python.

Cet exerciseur est utilisé pour des enseignements d'initiation à l'algorithmique et à la programmation, depuis 2016 par des étudiants en 1ère année de DUT informatique à Orléans (environ 120 étudiants chaque année) et depuis 2020 par des étudiants en 1ère année de Licence Informatique à Nouméa (environ 60 étudiants chaque année). Signalons que même si le contenu de ces enseignements considère les bases de l'algorithmique/programmation, le niveau de maîtrise et de pratique des étudiants intégrant ces formations est hétérogène. Certains étudiants ont déjà acquis des connaissances en programmation au Lycée ou en autodidactes, alors que d'autres n'ont jamais pratiqué la programmation.

L'exerciseur Python implémenté sur la plate-forme permet une évaluation automatique de *fonctions* Python à partir d'*entrées* pré-définies (cas de tests). Ainsi, lors du dépôt d'un exercice, l'enseignant doit fournir :

- un *énoncé* : consigne expliquant la tâche que doit réaliser la fonction,
- une *solution* : proposition d'un code solution,
- une série d'*entrées visibles* : cas de tests illustrant les résultats attendus de la fonction (affichés à la suite de l'énoncé) et servant à construire les *feedbacks*,

– une série d'*entrées invisibles* : cas de tests utilisés lors de l'évaluation du programme de l'étudiant mais qui ne sont pas portés à sa connaissance[5].

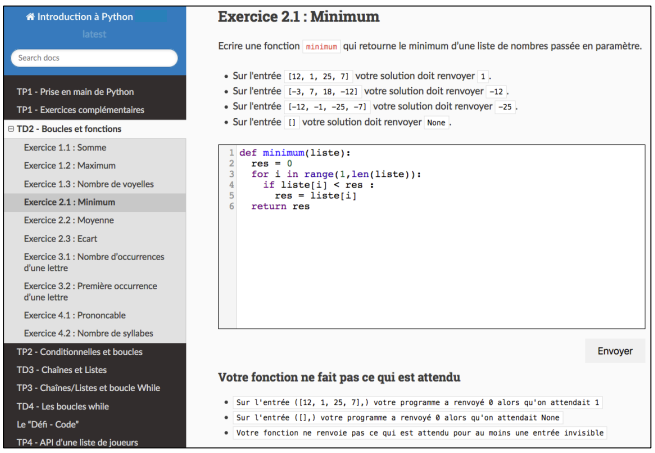

Fig. 1. Interface Web de la plate-forme d'entraînement à la programmation.

Dans l'interface présentée à l'étudiant (Figure 1), le menu à gauche de l'écran permet de naviguer entre les exercices. Pour chaque exercice, l'étudiant dispose de l'énoncé complété d'exemples basés sur les entrées visibles. Il dispose ensuite d'une zone de texte pour saisir son programme puis d'un bouton "Envoyer" pour déclencher l'évaluation automatique de sa proposition. Celle-ci consiste à comparer les résultats d'exécution du programme de l'étudiant d'une part et la solution de l'enseignant d'autre part sur chacune des entrées visibles et invisibles. Un feedback est alors affiché, explicitant notamment les erreurs (différences) observées sur les entrées visibles. Dans l'exemple illustré en Figure 1, l'étudiant a défini une fonction *minimum* syntaxiquement correcte ; cependant son algorithme présente une erreur dans l'initialisation de la variable `res` et ne gère pas le cas où la `liste` passée en paramètre est vide. Le feedback affiché au terme de l'évaluation oriente l'étudiant vers la compréhension de ses deux erreurs.

La plate-forme a été librement mise à disposition des enseignants de sorte que l'on observe une grande diversité dans les modalités pratiques de son intégration dans les enseignements. Certains enseignants l'ont adoptée comme environnement de programmation lors des travaux pratiques, d'autres ont également systématisé son usage mais seulement pour la validation, enfin une autre pra-

[5] L'ajout d'entrées invisibles permet d'éviter la validation d'un programme centré exclusivement sur les exemples portés à la connaissance de l'étudiant (entrées visibles).

tique a consisté à circonscrire l'usage de l'exerciseur comme un outil d'aide au travail personnel (en dehors des heures de formation). Depuis 2020, les traces d'utilisation de la plate-forme sont conservées à des fins d'exploitation scientifique et pédagogique. Nous conservons les programmes soumis pour évaluation[6] ainsi que les méta-données potentiellement utiles à l'analyse et la modélisation des processus d'apprentissage (identifiant étudiant, date/heure). La diversité d'usages a un impact significatif sur la nature des données récoltées.

3 Discussion sur la modalité pédagogique

Le dispositif pédagogique présenté offre des avantages indéniables parmi lesquels : d'une part, pour l'étudiant, une plus grande autonomie et une ludicisation de son apprentissage et d'autre part, pour l'enseignant, une délégation à la machine de la tâche répétitive de validation des programmes[7], profitable à sa pédagogie. Il nous paraît néanmoins essentiel de mener une discussion concernant les biais inhérents à ce dispositif, observés en situation réelle d'apprentissage. Les verrous pédagogiques qu'ils révèlent ne remettent pas en cause le recours à ces plate-formes mais constituent en revanche autant de pistes d'amélioration.

Tout d'abord, bien que l'outil numérique soit présenté à l'apprenant comme une solution d'apprentissage et non d'évaluation, le principe même de ludicisation *via* la validation automatique des solutions peut, chez certains apprenants, favoriser un comportement de type '*gamer*'. Il se met en quête de *trouver* la solution, y compris auprès d'autres apprenants, sans chercher à la *construire*.

Ensuite, il convient d'observer que le 'dialogue pédagogique' entre l'apprenant et la machine ne porte que sur des programmes complets. Contrairement à l'enseignant qui peut accompagner l'étudiant dans l'écriture de son programme, l'exerciseur n'intervient quant à lui qu'une fois un (premier) programme totalement écrit. Ceci précise les conditions dans lesquelles cet exerciseur doit être utilisé : en parallèle d'un accompagnement (synchrone) par l'enseignant et/ou limité à la réalisation de programmes courts ; cette dernière condition étant généralement vérifiée dans des enseignements d'initiation à la programmation.

L'observation des traces d'activités des étudiants sur la plate-forme met en évidence des situations de blocage, des étudiants réalisant parfois plusieurs dizaines de tentatives pour un même exercice[8] et parfois sans parvenir à une solution valide. Par exemple, pour un exercice on observe un étudiant qui a réalisé au total 98 tentatives - en deux périodes de temps (43 tentatives en moins d'une heure puis 55 tentatives sur 1h30 deux jours plus tard) - sans réussite finale. Au-delà du risque de découragement induit par ce comportement de type 'essai-erreur' improductif [8], il est plus inquiétant d'observer des situations de 'désapprentissage' pour lesquelles l'étudiant dégrade une première solution initialement bien conçue et proche d'une solution valide. L'analyse de sa séquence de

[6] Les traces sont conservées de façon anonyme et avec l'accord des utilisateurs
[7] assurant au passage une couverture plus large et systématique.
[8] jusqu'à 174 tentatives réalisées en 3 jours.

tentatives révèle sa propension à remettre en cause toutes notions préalablement acquises, qu'elles soient d'ordre conceptuelle ou purement syntaxique.

Enfin, lorsque l'enseignant analyse les programmes validés par la plate-forme, il constate une part significative de solutions non souhaitées (environ 1/3 pour notre expérience). Il peut s'agir de solutions fausses (∼15% des cas) révélant la nécessité de corriger les spécifications de l'exercice (solution enseignant, entrées visibles et/ou invisibles) mais aussi parfois de bonnes solutions que l'enseignant souhaiterait ne pas accepter (∼85% des cas) parce qu'il s'agit de mauvaises pratiques ou d'un contournement de la compétence ciblée par l'exercice (p.ex. l'utilisation d'une fonction de l'API au lieu du codage de l'algorithme associé).

Ces différents biais observés - *non construction des propositions*, *accompagnement limité*, *blocage*, *désapprentissage* et *validations non-souhaitées* - constituent, de notre point de vue, de véritables verrous pédagogiques qui, bien qu'ils existent dans des situations classiques d'apprentissage, sont susceptibles d'être accentués par l'usage d'outils numériques de type exerciseur. Une manière de surmonter ces difficultés consiste à replacer l'enseignant au centre du dispositif. En le considérant comme expert métier, il s'agit alors d'exploiter les données générées par ses apprenants pour l'aider à décider efficacement des interventions pédagogiques nécessaires. Dans tout processus d'aide à la décision, il est utile de proposer à l'expert une représentation/visualisation précise des données. Dans notre contexte, cela pose le problème de la représentation des programmes.

4 Apprentissage de représentations de programmes

L'utilisation des plates-formes d'entraînement à la programmation soulève donc un certain nombre de défis pédagogiques, mais elle ouvre aussi de nouvelles perspectives grâce à la multitude de données collectées. Par exemple, [2] exploitent des méthodes de fouille de texte pour prédire la justesse d'une réponse donnée par un étudiant (un programme Python). [11] utilisent quant à eux des réseaux de neurones pour propager automatiquement les *feedbacks* des enseignants. [12] utilisent les données collectées pour prédire les erreurs faites par les étudiants. De manière générale, les approches basées sur des représentations vectorielles, appelées encore *embeddings*, suscitent beaucoup d'intérêts [9].

Le principe de ces approches est de représenter chaque programme par un vecteur de réels (*embedding*). Cette représentation est ensuite utilisée en entrée d'un algorithme d'apprentissage pour construire un modèle prédictif. L'intérêt de ces *embeddings* est de projeter (ou 'plonger') tout un vocabulaire dans un espace vectoriel de faible dimension, tout en capturant des relations sémantiques complexes à partir de simples opérations sur ces vecteurs (p.ex. $vecteur(\text{"}king\text{"}) - vecteur(\text{"}man\text{"}) + vecteur(\text{"}woman\text{"}) \approx vecteur(\text{"}queen\text{"})$). Différentes approches ont été proposées pour construire des *embeddings* de programmes [1]. Toutefois, elles ont encore du mal à capturer totalement la sémantique sous-jacente, notamment les aspects fonctionnels et de style.

Face aux limites des approches existantes, nous avons proposé l'approche *code2aes2vec* exploitant instructions, structure du code et traces d'exécution,

afin de construire des *embeddings* plus fins. Nous présentons brièvement cette approche dans la suite de cette section, une description détaillée est présente dans [4]. Elle procède en deux étapes : 1) représentation des programmes sous forme de séquences d'exécution abstraites (étape *code*2*aes*); 2) utilisation de l'algorithme *doc*2*vec* [6] pour apprendre des *embeddings* de programmes à partir de ces séquences (étape *aes*2*vec*). Contrairement aux autres propositions, cette approche générique et non-supervisée intègre des éléments de fonctionnalités, de style et d'exécution. Ce dernier point est primordial dans notre contexte car il faut pouvoir différencier des programmes répondant à un même exercice (i.e. implémentant les mêmes fonctionnalités) mais de façons différentes (en terme de stratégie ou d'efficacité).

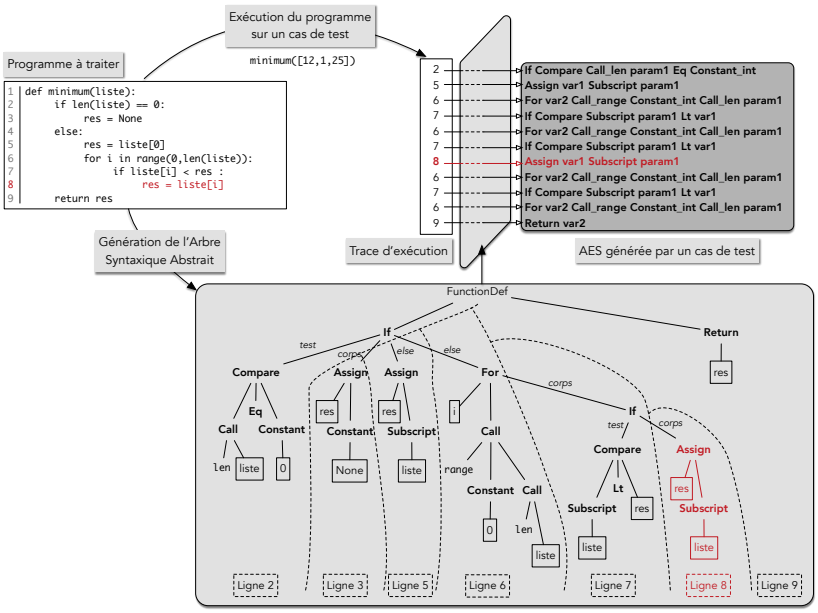

Fig. 2. Processus de construction des AES à partir d'un cas de test et d'un programme Python soumis en réponse à l'exercice 'rechercher le minimum dans une liste'.

L'étape *code*2*aes* consiste à traduire un programme sous forme de séquences d'exécution abstraites, ou AES (*Abstract Execution Sequence*). Chaque AES traduit la trace d'exécution du programme sur un cas de test (Figure 2), où chaque instruction est représentée par les "mots" du sous-arbre associé dans l'AST (parcours en profondeur). L'AES finale d'un programme est obtenue par concaténation des AES résultantes de chaque cas de test.

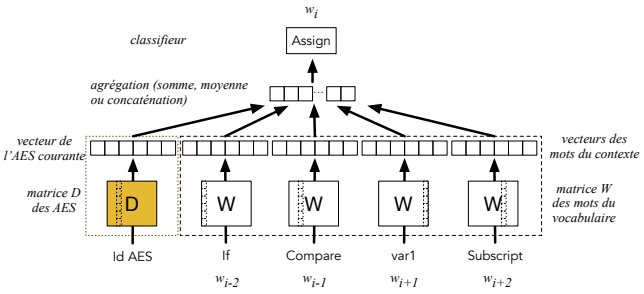

Fig. 3. Réseau de neurones utilisé pour prédire un mot w_i à partir de l'identifiant de son AES d'origine et de son contexte (deux mots précédents et deux mots suivants).

Ensuite, l'étape *aes2vec* exploite l'approche *doc2vec* [6] pour construire les *embeddings* des programmes à partir de leur AES. Pour cela, elle utilise un réseau de neurones (Figure 3). Ce réseau est entraîné à prédire le mot courant w_i d'une AES à partir de l'identifiant de l'AES, des mots précédents et des mots suivants. Chaque AES est associée à une colonne d'une matrice D, et chaque mot à une colonne d'une matrice W. Ces vecteurs sont ensuite agrégés par concaténation pour prédire le mot courant en utilisant un classifieur multi-classes du type *softmax*. Les paramètres sont mis à jour par descente de gradient stochastique. Une fois le modèle entraîné, la matrice D renferme les *embeddings* des programmes utilisés pour cet entraînement.

5 Exploitation des *embeddings* de programmes

Nous avons montré dans [4] que les *embeddings* appris avec *code2aes2vec* étaient suffisamment précis pour identifier la fonctionnalité d'un programme mais également pour distinguer des variantes de style. Ainsi, sur un corpus de 5690 programmes extrait des traces d'activités de nos étudiants sur 66 exercices, un simple classifieur supervisé (plus proches voisins ou SVM) permet d'identifier correctement l'exercice associé à un programme dans plus de 80% des cas.

Ces *embeddings* permettent aussi d'offrir à l'enseignant une visualisation globale des programmes de ses apprenants. L'utilisation d'un algorithme de réduction de dimensions adapté aux espaces d'*embeddings* (p.ex. une projection non-linéaire telle que proposée par la méthode t-SNE [7]), permet en effet de générer une carte en deux dimensions (ou plus si besoin) dans laquelle les programmes s'organisent selon leur typologie (syntaxe et fonctionnement). La Figure 4 illustre ce résultat sur l'un des exercices du corpus des 5690 programmes.

Pour aller plus loin, nous présentons dans cet article une preuve de concept montrant de façon concrète qu'il est possible d'exploiter les représentations issues

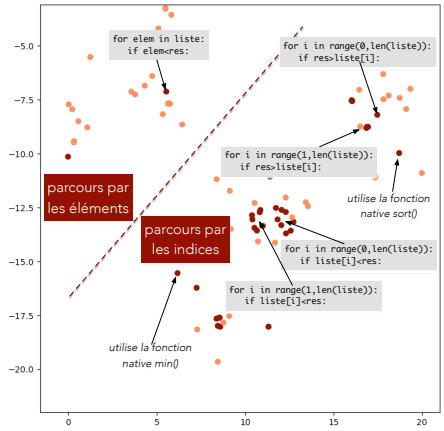

Fig. 4. Exemple de visualisation des programmes soumis par les étudiants pour l'exercice "rechercher le minimum dans une liste". Chaque point représente un programme soumis valide (en rouge) ou invalide (en rose).

de *code*2*aes*2*vec* pour aider l'enseignant à cibler efficacement ses interventions. Nous considérons plus particulièrement le problème - identifié en Section 3 - des *validations non-souhaitées*. Pour cela, nous avons demandé à des enseignants en informatique d'étiqueter les programmes validés par la plate-forme.

Parmi les 5690 programmes qui composent le corpus, 1304 ont été validés par la plate-forme (23%). Après suppression des doublons, l'ensemble S des exemples est circonscrit à 727 programmes. Chaque programme a reçu trois avis selon que l'enseignant juge sa validation non-souhaitable (il aurait voulu intervenir = exemple positif) ou souhaitable (il n'aurait pas jugé utile d'intervenir = exemple négatif). Une analyse rapide des étiquetages met en évidence à la fois :

- une expertise métier majoritairement partagée : les trois enseignants sont en parfait accord pour la majorité des exemples (54%),
- une différentiation pédagogique certes minoritaire mais réelle : sur 46% des exemples, la nécessité d'une intervention de l'enseignant n'est pas unanime.

Dans la suite de l'expérience nous avons considéré comme exemples positifs S^+ les programmes pour lesquels deux enseignants au moins jugent une intervention nécessaire, S^- étant de fait composé des programmes sur lesquels au moins deux d'entre eux ne jugent pas utile d'intervenir. S^+ représente 32% des exemples (234 programmes) et S^- renferme 68% des exemples (493 programmes). Nous avons ensuite cherché à apprendre un modèle de détection automatique des exemples positifs de sorte à utiliser ce modèle pour alerter l'enseignant sur des programmes susceptibles de nécessiter une intervention de sa part. S'agissant d'une tâche de classification supervisée (à deux classes), l'ensemble S a été

séparé en un ensemble d'apprentissage (80%) et un ensemble de test (20%). Pour représenter les programmes, nous avons utilisé un modèle d'*embeddings* appris avec l'algorithme *code*2*aes*2*vec* sur le corpus de 5690 programmes. Celui-ci a été paramétré de sorte à générer des vecteurs de représentation de taille 100. Nous avons considéré deux manières de représenter les programmes :

1. Sans recentrage : chaque programme $x \in S$ est représenté directement par son vecteur d'*embedding* $\phi(x) \in \mathbb{R}^{100}$ inféré par le modèle appris
2. Avec recentrage : les programmes sont recentrés par exercice tel que si s_i désigne la solution enseignant pour l'exercice $i \in \{1, \ldots, n\}$ et $E(x) : S \rightarrow \{1, \ldots, n\}$ la fonction associant chaque programme x à son exercice d'origine, la représentation $\phi'(x)$ d'un programme est donnée par $\phi(x) - \phi(s_{E(x)})$.

En considérant que l'espace des *embeddings* est organisé en groupes de programmes associés à un même exercice (hypothèse vérifiée dans [4]), l'opération de recentrage consiste à repositionner chacun des groupes *via* une translation de la solution enseignant pour l'exercice correspondant. Avec recentrage, la tâche de classification vise à apprendre un modèle générique indépendant de l'exercice.

Tableau 1. Taux de bonne classification calculé sur l'ensemble de test selon le type de représentation et de noyau considéré.

Classifieur	Sans recentrage	Avec recentrage
SVM linéaire	0.712	0.753
SVM polynomial	0.753	0.760
SVM RBF	0.719	**0.829**

Nous avons utilisé plusieurs algorithmes de classification supervisée et nous présentons en particulier les résultats obtenus par la méthode SVM consistant à apprendre des frontières de décision maximisant la marge entre les exemples des deux classes. La forme des frontières de décision diffère selon le type de noyau utilisé pour le SVM. Les modèles de classification ont été appris sur les données d'entraînement uniquement. Les taux de bonne classification obtenus sur les données de test montrent (Tableau 1) qu'il est possible de réaliser cette tâche de détection avec une précision supérieure à 80%, à condition d'une part de procéder effectivement au recentrage des données et d'autre part d'utiliser un noyau approprié. En l'occurrence, le noyau RBF semble adapté car il permet de construire des frontières correspondant à des zones convexes centrées sur des exemples (points de supports) positifs et positionnées à l'écart des exemples négatifs, situés quant à eux majoritairement au centre du nuage de points.

La Figure 5 confirme ces explications en présentant une visualisation des frontières de décision apprises sur une projection bi-dimensionnelle[9] (obtenue par l'algorithme t-SNE [7]) des *embeddings* de programmes.

[9] Les taux de bonne classification reportés sur la figure sont inférieurs à ceux du Tableau 1 du fait de cette projection.

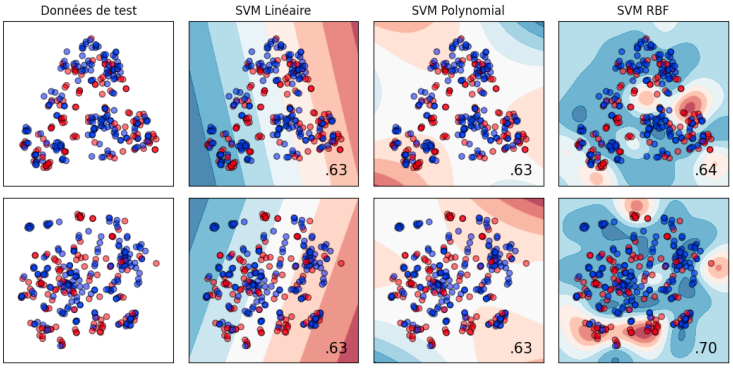

Fig. 5. Visualisation des frontières de décision apprises en fonction du type de noyau considéré et du type de représentation utilisé : sans recentrage des *embeddings* (ligne du haut) ou avec recentrage (ligne du bas). Positionnement des données de test : exemples positifs (rouge)/négatifs (bleu), et taux de bonne classification (coin inf. droit).

Au-delà du score de classification, il est utile pour l'enjeu applicatif, de détailler la validité intrinsèque du meilleur modèle appris (SVM RBF avec recentrage des *embeddings*). Le taux de bonne classification obtenu (0.829) correspond à la répartition suivante :

- 25 vrais positifs : programmes correctement identifiés comme nécessitant une intervention,
- 2 faux positifs : programmes identifiés à tort comme nécessitant une intervention,
- 23 faux négatifs : programmes nécessitant une intervention mais non détectés,
- 96 vrais négatifs : programmes ne nécessitant pas d'intervention et identifiés comme tel.

Il en résulte un modèle ayant une forte capacité à ne soumettre à l'enseignant que des programmes nécessitant son intervention (taux de spécificité de 0.925). Ce résultat très encourageant valide cette preuve de concept visant à assister efficacement l'enseignant à partir d'un apprentissage de représentations fines des programmes de ses apprenants.

6 Conclusion

Ce travail montre l'intérêt de l'apprentissage de représentations (embeddings) pour proposer de nouveaux outils de suivi à l'enseignant, et ainsi le remettre au cœur des plates-formes d'entraînement à la programmation. L'approche présentée s'appuie sur la construction d'*embeddings* sémantiquement riches pour

identifier des solutions justes mais nécessitant l'intervention de l'enseignant. La précision des résultats obtenus à ce niveau (82.9% de bonne classification avec seulement 2 faux positifs) illustre le potentiel de cette approche pour assister l'enseignant dans ses interventions (gain en temps et précision).

Les perspectives de ce travail sont nombreuses. Tout d'abord, la collecte de données complémentaires et introduisant d'autres langages de programmation permettrait d'évaluer la robustesse de l'approche et de tester d'autres scénarios pour détecter les solutions "valides mais non souhaitées". Il pourrait s'agir de construire un modèle par exercice (au lieu d'un modèle pour tous les exercices). De manière plus générale, ces *embeddings* de programmes ouvrent de nombreuses perspectives pour l'aide à l'enseignement. Ils pourraient être utilisés pour identifier des typologies d'erreurs, des solutions alternatives, ou des étudiants en situation de *blocage* voire de *désapprentissage* via l'analyse de leurs trajectoires.

References

1. Allamanis, M., Barr, E.T., Devanbu, P., Sutton, C.: A survey of machine learning for big code and naturalness. ACM Computing Surveys **51**(4), 1–37 (2018)
2. Azcona, D., Arora, P., Hsiao, I.H., Smeaton, A.: user2code2vec: Embeddings for profiling students based on distributional representations of source code. In: Proceedings of the 9th International Conference on Learning Analytics & Knowledge. pp. 86–95 (2019)
3. Broisin, J., Herouard, C.: Soutien à l'apprentissage de la programmation: conception et évaluation d'un indicateur sémantique. In: 9e Conference Environnements Informatiques pour l'Apprentissage Humain (EIAH 2019). pp. 235–246 (2019)
4. Cleuziou, G., Flouvat, F.: Apprentissage d'embeddings de codes pour l'enseignement de la programmation : une approche fondée sur l'analyse des traces d'exécution. In: 21ème conférence Extraction et Gestion des Connaissances (2021)
5. Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S.H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., et al.: Educational data mining and learning analytics in programming: Literature review and case studies. In: Proceedings of the 2015 ITiCSE on Working Group Reports, pp. 41–63 (2015)
6. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International conference on machine learning. pp. 1188–1196 (2014)
7. LJPvd, M., Hinton, G.: Visualizing high-dimensional data using t-sne. J Mach Learn Res **9**, 2579–2605 (2008)
8. Marzin-Janvier, P.: Des outils numériques pour apprendre les sciences (2020), https://hal.archives-ouvertes.fr/hal-03096594
9. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
10. Panigrahi, R., Srivastava, P.R., Sharma, D.: Online learning: Adoption, continuance, and learning outcome—a review of literature. International Journal of Information Management **43**, 1–14 (2018)
11. Piech, C., Huang, J., Nguyen, A., Phulsuksombati, M., Sahami, M., Guibas, L.: Learning program embeddings to propagate feedback on student code. In: Proceedings of the 32nd International Conference on Machine Learning. p. 1093–1102. ICML'15, JMLR.org (2015)
12. Wang, K., Singh, R., Su, Z.: Dynamic neural program embeddings for program repair. In: International Conference on Learning Representations (2018)