IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 30 December 2022, accepted 11 February 2023, date of publication 22 February 2023, date of current version 9 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3247189

RESEARCH ARTICLE

Toward Supporting CS1 Instructors and Learners With Fine-Grained Topic Detection in Online Judges

FILIPE DWAN PEREIRA[1], SAMUEL C. FONSECA[2], SANDRA WIKTOR[3], DAVID B. F. OLIVEIRA[2], ALEXANDRA I. CRISTEA[4], (Senior Member, IEEE), AILEEN BENEDICT[3], MOHAMMADALI FALLAHIAN[3], MOHSEN DORODCHI[3], LEANDRO S. G. CARVALHO[2], RAFAEL FERREIRA MELLO[5], AND ELAINE H. T. OLIVEIRA[2]

[1]Department of Computer Science, Federal University of Roraima, Boa Vista 69300-000, Brazil
[2]Institute of Computing, Federal University of Amazonas, Manaus 69077000, Brazil
[3]Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC 28223, USA
[4]Department of Computer Science, Durham University, DH1 3LE Durham, U.K.
[5]CESAR School, Centro de Estudos e Sistemas Avançados do Recife, Recife 50030-260, Brazil

Corresponding author: Filipe Dwan Pereira (filipedwan@gmail.com)

This work was supported in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico–Brazil–Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grant 308513/2020-7, in part by the Fundação de Amparoà Pesquisa do Estado do Amazonas (FAPEAM) under Grant 01.02.016301.02770/2021-63, in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior–Brazil (CAPES) under Grant 001, and in part by the Acuity Insights under the Alo Grant Program.

ABSTRACT Online judges (OJ) are a popular tool to support programming learning. However, one major issue with OJs is that problems are often put together without any associated meta-information that could, for example, be used to help classify problems. This meta-information could be extremely valuable to help users quickly find what types of problems they need most. To face this problem, several OJ administrators have recently begun manually annotating the topics of problems based on computer science-related subjects, such as dynamic programming, graphs, and data structures. Initially, these topics were used to support programming competitions and experienced learners. However, with OJs being increasingly used to support CS1 classes, such topic annotation needs to be extended to suit CS1 learners and instructors. In this work, for the first time, to the best of our knowledge, we propose and validate a predictive model that can automatically detect fine-grained topics of problems based on the CS1 syllabus. After experimenting with many shallow and deep learning models with different word representations based on cutting-edge NLP techniques, our best model is a CNN, achieving an F1-score of 88.9%. We then present how our model can be used for various applications, including (i) facilitating the search process of problems for CS1 learners and instructors and (ii) how it can be integrated into a system to recommend problems in OJs.

INDEX TERMS CS1 syllabus, natural language processing, topic detection, deep learning, programming autograder.

I. INTRODUCTION

Online Judges (OJ) are a special kind of massive open online course (MOOC) that provides a reliable, automatic, and instantaneous evaluation of an algorithm's source code sent by the learners [1], [2], [3], [4], [5]. The popularity of OJs are increasing in CS1 classes [1], [6], [7], due to their conveniences for both learners (e.g., automatic code correction) and instructors (e.g., workload reduction in providing feedback on students' code) [8], [9], [10], [11], [12]. Indeed, OJs facilitate the use of the Many Small Programs (MSP) approach [13], [14], [15], [16]. The MSP approach requires the students to solve many smaller programming assignments weekly or biweekly, instead of a traditional One

The associate editor coordinating the review of this manuscript and approving it for publication was Utku Kose.

Larger Program (OLP), in which the learners solve a unique and more extensive assignment in a short period of time. Notice that the MSP approach has brought many advantages for CS1 learners, including increased confidence, better exam performance, and reduced anxiety [17].

Despite the benefits of OJs for learning programming, recent studies [1], [6], [7] report that OJ systems need to be further improved to serve students, instructors, and administrators better. In this sense, one of the most significant issues of these systems is that the available problems are not organised based on meta-information that facilitates the users' searching process [1], [2], [12], [18]. More specifically, problems are often arranged in volumes without topics classification. The main reason is that this annotation task is quite labour-intensive and requires the availability of experts.

In light of this, a few studies [2], [18], [19], [20] have recently proposed methods for the automatic categorisation of problems in OJ systems. However, such methods categorise problems into non-granular levels, such as *data structure*, *beginner*, *paradigms*, and so on. Such granularity in the annotation is particularly useful for online judge administrators and experienced students. To illustrate, administrators can use such methods to automate the annotation process, even considering the error component - as these methods are based on Machine Learning (ML) techniques, which are not 100% accurate. Students can use the automatically annotated topics to find desirable problems in generic categories, which is far more efficient than navigating multiple volumes [1], [18], [19].

Nonetheless, thinking of CS1 students and instructors, this generic granularity in categorisation is not enough. To illustrate, CS1 instructors who use an MSP approach tend to look for problems of a simple conditional structure (if-then-else), nested conditional structure, or loops to compose the small assignments lists instead of generic topics such as ''beginner'', or advanced topics such as ''computational geometry. In addition, for novice students to be able to choose problems compatible with their level of knowledge in an unassisted way (without the intervention of the instructor/tutor), a granular categorisation of problems, compatible with the CS1 course syllabus, is crucial. That is, a student who is learning to manipulate vectors would benefit from doing a search for problems in the vector category,[1] instead of looking for this type of problem in a single category called ''beginner''. However, to the best of our knowledge, there is no method for automatically classifying problems available in OJ at this level of granularity, i.e., that may further suit CS1 students, instructors, and administrators. In this sense, we formulate the following research question:

> *RQ*: *How can we provide a method that can be used to support instructors, students, and OJ administrators in classifying problem topics according to the CS1 syllabus?*

[1] We use the words *category* and *topic* with the same meaning in this work for the sake of simplicity.

To answer this research question, our hypothesis is that, if an expert human (e.g., an instructor) reading a problem statement can detect the topic, then Natural Language Processing (NLP) techniques can be employed to represent the statement data and be used as inputs in machine learning algorithms to automatically classify the topics based on the CS1 syllabus. Briefly, the main contributions of this work are:

- proposing and validating different combinations of ML and NLP techniques to perform topic detection based on the CS1 syllabus;
- providing discussions of applications and implications of employment of our best predictive model;
- providing a method to help OJ systems better assist CS1 students and instructors, and novice learners.

II. BACKGROUND

A. ONLINE JUDGES

Online judges (OJs) are systems designed for the reliable assessment of algorithm source code submitted by learners, which is next compiled and tested in a homogeneous environment [1], [5], [6], [15], [18]. OJs have been used since at least 1961, where they were developed at Stanford University to help evaluate students' written programs [1]. OJs were also used in programming contests, where participants were asked to solve as many algorithmic problems as they could for the duration of the contest, as an automatic evaluator of participants solutions.

The evaluation procedure of an OJ, as defined by [1], consisted of three steps: (1) submission, (2) assessment, and (3) scoring. In the *submission* step, the submitted code was compiled and verified to have been successfully executed in the evaluation environment. Once it could be executed, the submission was *assessed*, based on a problem-specific set of test cases. The final *score* was computed as an average from all test cases.

The use of online judges in education to evaluate students' programs has brought many advantages [1], [5], [6], [15], [17], [18]. First, the instructor could assess the correctness of students' submissions more efficiently and accurately. For students, they received almost instantaneous feedback on their submissions, as to whether or not it was correct. Some OJs also provide the percentage of test cases that the student code passed.

Importantly, OJs provide many problems for students and instructors [1], [6], [12], [18]. The former can use the system for to develop their skills in specific computer science topics such as graphs, dynamic programming, and in beginning concepts, such as vectors, matrices and so forth. The latter can select problems from OJs to compose assignments and exams in different level (from beginner to more advanced) of programming courses. However, overall, those systems do not provide any kind of topic annotation for the problems, which make the searching process challenging both for students and for instructors.

B. MANY SMALL PROGRAMS APPROACH

Many Small Programs (MSP) is an approach which requires the students to solve many smaller programming assignments weekly or biweekly, instead of a traditional One Larger Program (OLP), in which the learners solved a unique larger assignment in a short period of time. The periodicity (weekly/biweekly) could be determined by the CS1 instructors or via an institutional decision (at department level) [13]. Recent studies suggest that the MSP approach can help students feel less stressed, more confident, and make them achieve higher levels of satisfaction and greater performance [14], [17]. Indeed, based on recent MSP results, an increasing number of computer science departments have adopted this approach in CS1 classes [15], [16], [17], [20], [21], [22].

The MSP approach is enabled by OJ systems, as they allow easy creation of assignments that give students immediate feedback about the correctness of their code solutions.

III. RELATED WORK

One way to classify programming problem topics is by asking instructors or experts to perform this task manually. However, this process requires the availability of these annotators and takes a considerable amount of time and human effort. Notice that this manual annotation is not scalable. Scalability is crucial for systems like OJs since new questions are being registered constantly [23], [24], [25]. Thus, it is important to propose strategies to automate the organisation and annotation of OJ questions with useful meta-information. Indeed, automatic classification methods to detect the problems' categories in OJ are essential for instructors and students of computer science majors [1], [2], [12]. Moreover, a potential automation in the annotation would bring many benefits to OJ administrators, since, to the best of our knowledge, annotation is performed manually in the few OJs in which problems are labelled with topics. Nonetheless, extracting knowledge concepts or topics from programming exercises is barely explored by the Computation Education literature [26]. The few works that tackled this problem will be discussed in this section.

Reference [18] proposed a method for the automatic detection of problem topics based on sequences of timestamped attempts and correctness from OJ students. Using machine learning techniques, [18] detected the following advanced topics related to computer science: dynamic programming, palindromes, geometry, tricky problem, hardest problem, data structures, number theory, graph theory problem for beginners, game, unusual problem, string algorithms. The authors used student data from 940 problems collected from three different OJs. As the students' timestamps, attempts and correctness might vary too much in different OJs, authors opted to train and test the ML models with data from the same OJ. As a result, the authors achieved a modest F1-score of 75.1%, 64.4%, and 72.4%, for each dataset. Still, a major limitation of the method proposed by [18] was that new problems enrolled in the OJ could not be annotated by their predictive method, since there were no attempts of resolution made by the students available for the new questions. In other words, problems needed to be solved by at least a couple of students to render the data (timestamped attempts and correctness) needed to feed the ML model and, thus, allow the model to make a decision. As such, recent works used the text extracted from problem descriptions to feed machine learning models. This would allow the categorisation of new problems in OJs, as a move towards scalability. Moreover, NLP techniques have been achieving prominent results in many other tasks [27], [28].

In this sense, [29] extracted text-data on problems' description using NLP techniques and then used shallow machine learning models to perform a multi-classification task with 5 topics (data structures, dynamic programming, greedy, ad-hoc, and math). Using a total of 1709 problems, [29] achieved modest F1-scores, ranging from 19.2% to 62.2%. Similarly, [19] also proposed an NLP method using the problem descriptions to automatically categorise OJ problems, using 4620 problems. In total, they used 7 topics (beginner, ad-hoc, strings, data structure, mathematics, paradigms, graph, computational geometry), based on the topics covered in the International Collegiate Programming Contest (ICPC). As a result, they achieved an F1-score of 86%. All the authors ([18], [19], [29]) cited in this section that performed topic detection have utilised the topic-annotation provided by the OJ system by performing web scrapping.

Note that the granularity of the topics used by the studies cited in this section is not ideal neither for novice programmers, nor for CS1 students and instructors. To illustrate, [19] used a category called "beginner", which ranges from exercises using basic sequential structures to exercises using vectors and matrices. On the other hand, other studies focused only on advanced categories, such as paradigms, computational geometry, and so on. As such, there is a gap for novice students and CS1 students, forcing them to do an exhaustive problem search for their respective levels. For example, for a student who is learning simple "if-then-else conditional structure", it would be ideal to find problems for this category quickly. Moreover, CS1 instructors also struggle to find appropriate problems to compose assignment lists, mainly for the ones that use MSP approach. Thus, CS1 instructors could benefit from an outline annotation based on the CS1 syllabus.

Indeed, in recent works [30] and [12] about recommending problems in OJs to CS1 students and instructors, the authors explained that a recommender could be useful for students by allowing them to select appropriate problems unassisted, and for instructors by allowing them to select problems to compose lists of exercises and tests. Nonetheless, the recommender method has a limitation related to problem topics. The authors of both works explained that it was not possible to make a more fine-grained recommendation in which users would choose a topic of interest because the problem topics have not been annotated based on the CS1 syllabus.

As such, we for the first time, to the best of our knowledge, employ an NLP pipeline combined with deep and shallow machine learning algorithms to automatically detect problem topics based on the CS1 syllabus. To do this, we use problem descriptions as input into the NLP and machine learning algorithms. We opted for the problem description rather than student data, since using problem descriptions allows the model to classify problems that have not yet been solved by any students. Finally, it should be noted that we have made our database available with this fine-grained topic annotation, so that other studies can employ our pipeline and data on online judges containing questions without such annotations.

IV. EDUCATIONAL CONTEXT

The data for this study was collected from an educational setting, where CS1 is offered as a pre-requisite to students taking any of 15 available non-CS courses segregated in five majors fields: Geology, Physics, Engineering, Mathematics, and Statistic. Although CS1 is mandatory for these courses, there is a lack of motivation on the part of the students, as some of them cannot see the usefulness of programming for their professional careers. Thus, approaching advanced content and setting a very high expectation around the content to be taught can be frustrating for instructors, since they might not be able to teach everything they would like, and students, as they might not achieve what is expected of them.

Based on this premise of maintaining a realistic expectation of what should be taught, and on empirical observations established over years of teaching the CS1 course for non-CS students at Federal University of Amazonas (UFAM), a group of instructors and researchers established a standard methodology to be used in all CS1 classes. The methodology is based on the pedagogical MSP approach [13].

In this methodology, the topics covered throughout the course, taught sequentially, are the following: Variables and Sequential Structure, Simple Conditional Structure (if-then-else for short), Conditional Structure Nested (if-then-else (nested) for short), Repeating by Condition (while-loop for short), Vectors and Strings, Repeating Structure by Count (for-loop for short), Matrices. Using this methodology, classes are organised as follows:

- Two starter classes to teach input/output commands, variables and online judge familiarisation.
- In subsequent classes, two weeks are set aside to teach each topic. In total, 4 classes are taught over the two weeks. The four classes are distributed as follows:
  - a theoretical class,
  - two practical classes - two laboratories,
  - one exam.

It is noteworthy that this teaching methodology was designed so that instructors could teach a piece of content and then verify students' understanding of it through practical work (practical classes and the exam). Note that one effect of using this methodology is to keep topics simple and short, which can be taught in one concept class and tested in both labs and subsequent exams.

This explains, for example, separating *if-then-else* from *if-then-else (nested)* in distinct topics. In fact, based on our classroom experience, we know that most students would not 'digest' the use of nested and simple conditions in a single classroom event. Similarly, teaching the use of while-loops and for-loops in a single class can make the student leader only one of the two, or even confuse them.

Note that for repetition by counting, the number of repetitions of a given procedure must be explicit in the statement of the problem, whereas for repetition by condition, the condition must be explicit. An example of repeating by count would be, e.g., repeating a procedure six times, while repeating by condition would be reading and accumulating numeric values until the value of -1 is read. By teaching them separately, it is expected that students will be able to develop both skills (*for-loop* and *while-loop*).

Finally, to support instructors and allow immediate feedback to students, an OJ system was used as an instrument to perform the automatic correction of student codes. Both assignments and exams were made available to students through the OJ system.

V. RESEARCH DESIGN

A. DATA COLLECTION

In this research, we extracted data[2] from an online judge system called CodeBench[3] due to convenience sampling. This system is used as a tool (more detail in Section VI-A) to support programming classes at the Federal University of Amazonas. In our case, only questions solved by CS1 students were considered.

In the CS1 classes, students both solved an assignment list that preceded an exam and took the actual exam in this system. Each assignment contained an average of 10 questions, whilst each exam comprised 2 questions. Thus, each assignment, together with the successor exam, formed a *session* for the purposes of this research. In total, 7 sessions $(s1, s2, \ldots, s7)$ were carried out for each course. Students used the Python programming language.

Importantly, each session is associated with a specific topic from the CS1 curriculum (see section IV). Moreover, before the 7 sessions, students have a first week to get used to the Python programming language; this involves completing simple problems that require the use of prints and input commands. We call this introductory week the adaptation moment *a0* (see Table 1).

To create the assignment lists for each session, the instructors selected the problems from the database of questions. There was a total of 1045 questions related to the CS1 syllabus available. For the exams, instructors are required to create 10 different versions of the same exam, in order to diminish plagiarism.

[2]github.com/filipedwan/IEEE-Access-CS1-Topic-Prediction
[3]http://codebench.icomp.ufam.edu.br/

In an effort to facilitate the process of selecting the problems to compose assignments and exams for the CS1 classes, the group of 12 instructors from UFAM manually annotated the topic of all problems based on the CS1 curriculum. Using this annotation, the instructors can filter the problems from an specific topic to create the assignments and exams of specific sessions.

All annotators have previous experience with CS1 teaching. Since there are more than a thousand problems available on CodeBench, the annotation scheme of each problem is performed by a single instructor. So that each instructor annotated a portion of the questions. In addition, before using the problems in their assignments or exams, the CS1 instructors could relabel the problems' topics if necessary.

B. PROBLEM DEFINITION

We are interested in classifying programming problems in terms of topics based on CS1 syllabus. A problem that can be solved with a vector, for example, can be classified as pertaining to the 'vector' topic. We used the topics presented in section IV in our target variable since we used the data collected from this educational context, which uses the MLP approach, which is popular in CS1 courses.

As such, we model this task of classification as a multi-classification problem, in which the classifier must predict one out of eight topics for each problem description. Thus, we use the problem statements as input. Using this input, the classifier must provide a predicted topic. To do so, we will first apply NLP techniques over the problems statements, to then feed the numerical features into the ML algorithms.

Table 1 shows the descriptions for each topic (where *a0* depicts the period when students get used with the CodeBench, and the number of problems per topic). Moreover, to clarify the concepts covered in each topic, in Table 1 we provide an example of code illustrating a typical answer to a question on that topic.

Additionally, we performed two adaptations in the way instructors annotate the topics. First, as we aim the resulting algorithm to also detect the topic "input and print", even if it was taught in the pre-course week, we added it to the table. Moreover, we divided *Vectors and Strings* into two different topics, since these topics have different nuances, and instructors and students probably would like to search for them separately.

Notice that one problem can fall in only one topic or multiple topics. The way the annotators employed to prioritise topics if the problem belongs to multiple topics takes into account the MSP pedagogical approach and the educational settings presented in section IV. That way, if a problem has elements of more than one topic (e.g., if-then-else and for-loop), the annotators considered only the most complex topic (repeating structures). This is because learning in the MSP approach is cumulative, so if the student is in the week where for-loop is learned, it is assumed that he already knows how to deal with if-then-else. As such, there is no need for both labels (if-then-else and for-loop), only for the most complex topic.

VI. METHODOLOGY

This section presents the methodology we use to automatically detect the topics of programming questions based on their descriptions.

A. TOOL

We use the online judge system Codebench, a self-devised OJ implemented and maintained by one of the authors. Figure 1 shows a screenshot of its interface. The description of the problem can be seen in the left upper corner. A code solution written by one of the authors (as an illustration) is also provided on the right side of the figure. Tips and input/output examples were not used in the machine learning algorithms, due to two reasons: i) we observed that they typically do not contain key context or terms to detect the problem topic ii) many problems do not provide tips. As we wanted our approach to be widely generalisable, we opted to not use features which would not be available across OJs.

In the next subsections, we show how we employ this text-data, to feed the machine learning algorithms responsible to detect the question topics.

B. PREPROCESSING THE PROBLEM DESCRIPTIONS

Since there is a higher abundance of tools and frameworks for natural language processing techniques to deal with text written in English, we use first the Googletrans API[4] to translate the descriptions of the problems from Portuguese to English.

In addition, we applied the following widely used steps [20], [27], [31] to pre-process and clean the text-data and prepare it for the ML algorithms:

- Step 1: Removal of HTML tags from the text.
- Step 2: Replacing line breaks with single spaces and removal of punctuation symbols that potentially do not provide information relevant to the detection of problem topics.
- Step 3: Replacement of numeric values by the size of their magnitudes, using the # character (e.g., 112 $\rightarrow$ ###, 12 $\rightarrow$ ##).
- Step 4: Tokenisation, that is, breaking each problem description into a vector of single words.
- step 5: Stop words removal and lemmatisation.

To perform steps 1, 2 and 3 we created our own script and for the steps 4 and 5 we used the spacy library.[5]

C. TEXT REPRESENTATION

Machine learning algorithms receive text data as a numeric representation. (see Keras Tokenizer function[6]). Furthermore, the number of columns in the feature matrix is expected

[4]py-googletrans.readthedocs.io/en/latest/
[5]spacy.io/
[6]keras.io/preprocessing/text/

TABLE 1. Description of CodeBench topics, number of problems, and an example of code implementation for each topic.

	Topic	Description	Example of implementation	N
a0	print and input	reading a single variable and use of the print command	`print(20*35)`	19

s1	sequential structure	arithmetic operations and use of variables	`from math import *`
			`r = float(input())`
			`c = pi * r ^ 2`
			`print('%.2f', c)`
				157

s2	if-then-else	conditional structure	`from math import *`
			`r = float(input())`
			`c = pi * r ^ 2`
			`if r > 10:`
			`    print('big␣circle')`
			`else:`
			`    print('small␣circle')`
				136

s3	if-then-else (nested)	nested conditionals structure	`from math import *`
			`a = float(input())`
			`b = float(input())`
			`c = float(input())`
			`if (a > 0 and b > 0 and c > 0):`
			`    if ((a < b + c) and (b < a + c) and (c < a + b)):`
			`        s = (a + b + c) / 2.0`
			`        area = sqrt(s * (s - a) * (s - b) * (s - c))`
			`        print("Area:", area)`
			`    else:`
			`        print("Invalid␣area")`
			`else:`
			`    print("Invalid␣area")`
				161

s4	while-loop	repetition structure by condition - loops using the structure while	`num = int(input())`
			`while (num != -1):`
			`    if (num % 2 == 0):`
			`        print("even")`
			`    else:`
			`        print("odd")`
			`    num = int(input())`
				114

s5	for-loop	repetition structure by count - loops using the structure for	`import numpy as np`
			`n = int(input())`
			`print(np.ones(n, dtype=int))`
				117

s6	strings	operations on strings	`string = input()`
			`print(string.upper())`
			`print(string*500)`
				47

s6	vectors	operations on uni-dimensional vectors	`data = eval(str(input()))`
			`speed_limit = data[0]`
			`i = 1`
			`accumulator = 0`
			`minimum_limit = speed_limit + (speed_limit * 0.2)`
			`maximum_limit = speed_limit + (speed_limit * 0.5)`
			`for x in data[1:]:`
			`    if x > minimum_limit and x < maximum_limit:`
			`        print(i)`
			`        accumulator = accumulator + 1`
			`    i = i + 1`
			`print(accumulator)`
				160

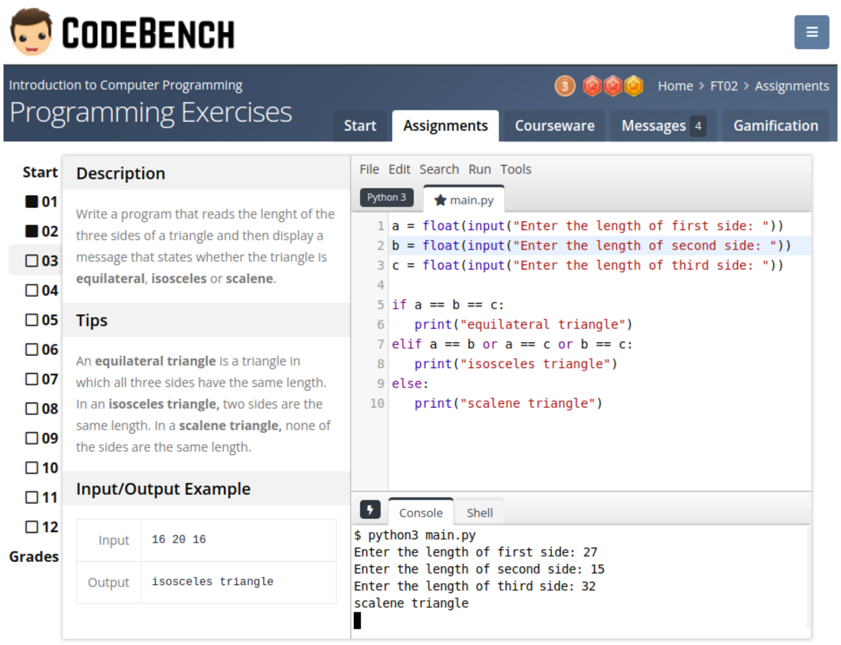
s7	matrices	operations on bi-dimensional matrices	`from numpy import *`
			`m = eval(str(input()))`
			`n = eval(str(input()))`
			`matriz = np.zeros((m,n),dtype=int)`
			`print(matriz)`
				134

to be the same for all rows. In other words, the same number of tokens is required for each sentence (row), even if the length of the question descriptions is different. Thus, a first measure we have taken was to complete with zeros (padding) the tokens of the lines whose number of tokens was smaller than that of the longest sentence. Tokenisation and padding are standard procedures in NLP. We chose to use the tokenisation and padding function of the Keras text preprocessing library[7]).

Additionally, the tokens can be represented as vectors extracted through prediction techniques with shallow models (e.g., googleNews-Vectors) or through counting techniques. In this study, we tested different state-of-the-art techniques for representing tokens as vectors. The first one we tried was googleNews-Vectors (W2V).[8] In addition, we also employed Glove [32] word embeddings. For the deep learning models, we used layers of the network itself to represent the word embeddings, as explained in Keras library.[9] The intention was to test various combinations of textual representation associated with various classifiers in order to find a pipeline that obtained cutting-edge results for our research problem.

It is worth mentioning that we also evaluated traditional feature extraction techniques such as Bag of Words (BoW), Term Frequency–Inverse Document Frequency (TF-IDF) and Latent Dirichlet Allocation (LDA). However, we decide not to keep these techniques in our pipeline for two reasons. The first one is theoretical, since these techniques do not capture semantics information or correlation between words, which is important in our context because problem statements from the same category potentially either share semantics or are correlated. The second reason is practical, since we achieved poor results for all tasks using BoW, LDA and TF-IDF representation and, hence, adding them would only increase the number of combinations and complexity in our pipeline, without any gain in terms of performance. Moreover, such increase in the

[7]keras.io/preprocessing/text/
[8]code.google.com/archive/p/word2vec/
[9]keras.io/



Description

Write a program that reads the lenght of the three sides of a triangle and then display a message that states whether the triangle is **equilateral**, **isosceles** or **scalene**.

Tips

An **equilateral triangle** is a triangle in which all three sides have the same length. In an **isosceles triangle,** two sides are the same length. In a **scalene triangle,** none of the sides are the same length.

Input/Output Example

Input	16 20 16
Output	isosceles triangle

```python
a = float(input("Enter the length of first side: "))
b = float(input("Enter the length of second side: "))
c = float(input("Enter the length of third side: "))

if a == b == c:
    print("equilateral triangle")
elif a == b or a == c or b == c:
    print("isosceles triangle")
else:
    print("scalene triangle")
```

```
$ python3 main.py
Enter the length of first side: 27
Enter the length of second side: 15
Enter the length of third side: 32
scalene triangle
```

FIGURE 1. CodeBench interface.

number of combinations (practical implication) would make it more difficult for other researchers to replicate our work.

D. OVERSAMPLING ON MINORITY CLASSES BY PARAPHRASING THE PROBLEMS' DESCRIPTIONS

To better adjust the class distribution in a move to reduce the bias yielded by the unbalanced nature of our dataset, we opted to use an oversampling technique. The technique consists in creating contextual paraphrases (artificial instances) of the statements from the training set. To do so, we used a library called NLP augmentation [33], recommended by previous works [34], [35], [36]. To create syntactic statements in our training set, we used word embeddings from the pre-trained cutting-edge model BERT. We used the k-nearest-neighbour and cosine similarity to find similar words for replacement.

However, similar to previous works [19], [20], we, at most, duplicate the number of instances on the minority classes from the training set to avoid too much artificial information, rendering the data as non-representative. The upper bound was defined based on the number of instances present on the topic of the majority class ($N = 160$). To illustrate, assume the majority class is x, which contains x' problems. Also assume that, in another given topic y, the number of problems are y'. If $y' > x'/2$, then we create only $x' - y'$ artificial instances for the class y, capping the number of instances in topic y at $||x||$. Otherwise, we duplicate y'.

E. CLASSIFICATION PROCESS

In this study we represented the problem as a multiclassification problem. Thus, the last layer of all deep neural network models was a softmax. The deep neural network models used for the classification are: i) a Recurrent Neural Network (RNN), ii) a Convolutional Neural Network (CNN), iii) a hybrid RNN and CNN (**RNN+CNN**) and iv) a *Bidirectional Encoder Representations from Transformers* (**BERT**). As for the network topology, the RNN had an embedding layer configured with the input length of 300; followed by a *Long Term and Short term memory* layer with 64 nodes; followed by a global max pooling 1D layer; one dense layers with 64 nodes; and a softmax layer. The CNN had an embedding layer configured with the input length of 300; followed by a convolutional layer with relu as the activation function a filter of 128 and kernel size of 64; followed by a global max pooling 1D layer, followed by a dense layer with 64 nodes; and a softmax layer. For the RNN+CNN, we stacked the aforementioned (RNN and CNN) deep learning models. For BERT, we used the default parameters of the "bert-base-uncased" pre-trained model.

Furthermore, as recommended by prior works [19], [27], we also compared the deep learning models with the following shallow models: i) Random Forest Classifier (RFC) ii) Decision Tree (DT), iii) Extremely Randomized Tree Classifier (ETC), iv) Support Vector Machine (SVM), v) vi) Gradient Boosting Classifier (GBC), vii) Naive Bayes (GNB), viii) XGBoost (XGB).

F. EVALUATING THE PREDICTIVE MODELS

To evaluate the performance of the predictive models, it is important to choose a main performance measure. *Accuracy*, *precision*, *recall* and *F1-score* are some of the most used measures. The *accuracy* is only the percentage of hits of the model, and is not recommended for scenarios where

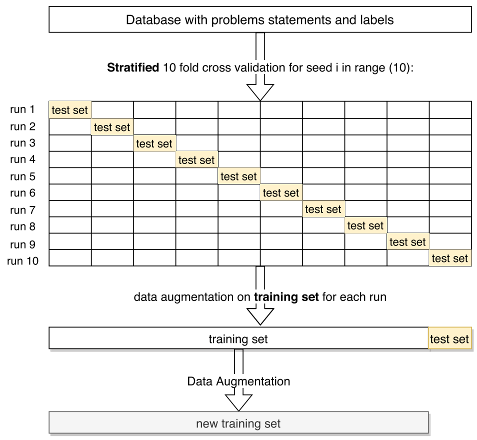

Database with problems statements and labels
Stratified 10 fold cross validation for seed i in range (10):
run 1
run 2
run 3
run 4
run 5
run 6
run 7
run 8
run 9
run 10
test set
data augmentation on training set for each run
training set
test set
Data Augmentation
new training set

FIGURE 2. Illustration of how we performed data augmentation on the training set.

the database is unbalanced. *Precision* gives an idea of how effective a model is at predicting instances of a class, but a high *precision* value does not mean good correctness. The *recall* measures how often the model finds examples of a class without providing precision in the classification.

In order to have a balance between precision and recall, we adopted the *F1-score* as the main metric to measure the performance of the classification models. This metric combines *precision* and *recall* by calculating a harmonic average in order to bring a single value that indicates the overall quality of the model. In addition to the *F1-score* metric, we also show a confusion matrix and an error matrix. In this way, errors can be analysed more in-depth.

For evaluation, each classifier was independently trained through cross-validation to minimise the possibility of overfitting the data. As our data has a high class imbalance, we divided the dataset into training and testing within the folds in a stratified way. We performed the stratified 10 fold cross-validation 10 times, using 10 different seeds (ranging from 0 to 99) to shuffle the data, due to statistical constraints. Thus, we obtained 100 (10×10) different results, i.e. one result for each test set. Finally, we average the computed performance over each test set. Figure 2 illustrates the process of data augmentation and validation. We performed that process of evaluation employing the *StratifiedKFold*[10] from scikit-learn [37]. It is worth noting that we performed data augmentation only on the training sets.

VII. RESULTS AND DISCUSSION

We built a total of 36 predictive models. Figure 3 illustrates all the results obtained by all models applied in this research. From this figure, as expected, in relation to the shallow methods, we observed that ML algorithms based on ensembles obtained better results than non-ensembles models (e.g. DT and NB). We also observed a slight advantage regarding word representation using W2V when compared to GLOVE, which is consistent with NLP literature [38], [39].

Another observation we identified is that the deep learning models, in general, outperformed the shallow models, as expected. We performed the Mann-Whitney hypothesis statistical test, and the model with the highest performance is the CNN model using paraphrasing, with an F1-score of $\approx$89% ($p - value < 0.05/36$ - Bonferroni correction). Although it is not usual for CNN outperforms BERT, previous studies reported that for a dataset similar to the one used in this work, this result is not unique [40], [41].

We can also notice that paraphrasing has not improved the *F1-score* (weighted) in most of the models. Indeed, paraphrasing boost significantly ($p - value < 0.05/2$ - Bonferroni Correction) only the results of the BERT classifier. *BERT* achieved an F1-score of $\approx$83%, without paraphrasing, whereas with paraphrasing, the model achieved $\approx$86%, an increase of 3%. These results are contradictory to previous studies [19], [20], [31], which also use contextual paraphrasing with BERT as a step to augment the training data. However, it is important to highlight that our dataset is smaller in terms of the number of instances in total and in each class, which probably affected the outcome of BERT, causing overfitting or unstable learning [28].

In addition, in Figure 4 we can analyse the performance of our best predictive model on each topic. The leftmost figure shows the confusion matrix obtained by our CNN, where the lines represent the real values, whilst the columns depict the estimated values. Our model achieved a recall of over 75% across all topics. In some topics, our model achieved a recall of more than 90% (e.g. if-then-else (nested), matrices, and vectors). We are not able to compare it fairly with the literature, since we did not find any work that performed this task of classification based on CS1 topics. However, compared to similar works which also performed topic classification in OJ systems, we can notice that our results are satisfactory. Indeed, [19] achieved the highest F1-score for a similar task, which is inferior to our CNN performance.

Next, analyse the model's errors in more detail. In order to compare errors proportionately, we divide each value in the confusion matrix by the number of instances in the relevant class. Then we fill the major diagonal with zeros, allowing us to focus solely on the misclassifications. In the rightmost image in Figure 4, we can see the resulting error matrix. Here, the darkest regions of the matrix show the places where the classifier made more errors. The misclassifications are not perfectly symmetrical. To illustrate, there are more *if-then-else* problems misclassified as *if-then-else (nested)*, than the reciprocal. Moreover, the *sequential structure* was mainly confused with *print and input* and *if-then-else*, which makes sense, since the *sequential structure* concept is present in almost all of the other topics, making it challenging for multi-classification. This occurs in other classes as well, such as *if-then-else*, which can be contained in problems of repetition, matrices and so forth. Another likely source of misclassification is that of *for-loops* being estimated as *vectors*. A possible

[10]scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

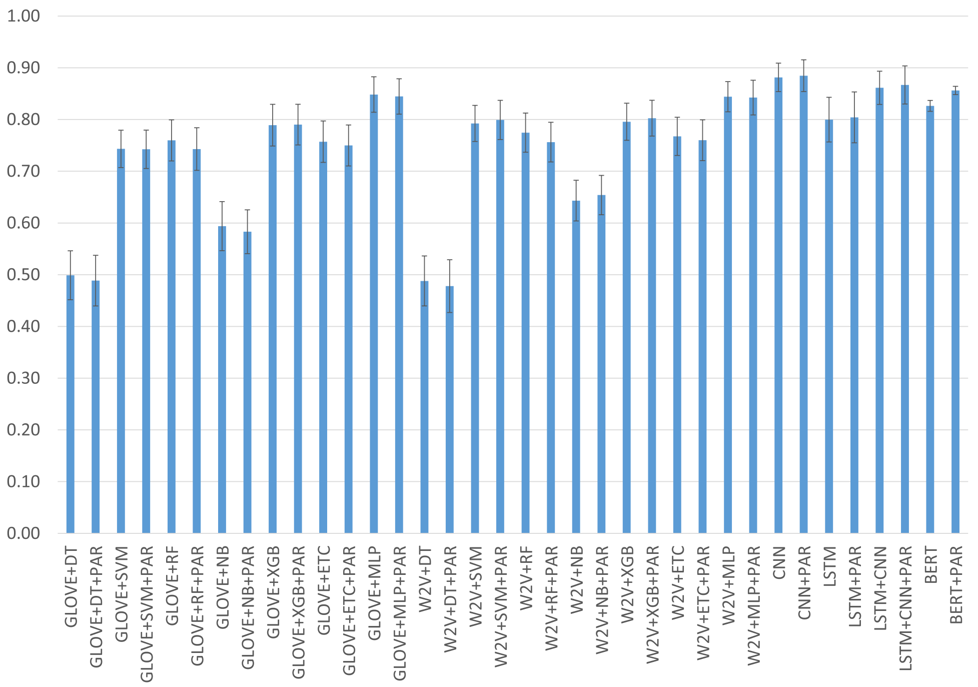


FIGURE 3. F1-score for CS1 topic prediction. We plot the weighted average F1-score, since our dataset is imbalanced and this metric considers the proportion of each class for calculation. In this figure, PAR refers to the use of paraphrasing in the training set.

justification is that a vector problem might be solved using *for-loops*.

VIII. ADDRESSING THE RESEARCH QUESTION

We can state that our CNN achieved results that support our RQ (*How can we provide a method that can be used to support instructors, students, and OJ administrators in classifying problem topics according to the CS1 syllabus?*). Indeed, we believe that our method can be employed in an online judge system to support CS1 students and instructors. As we have shown, OJs are popular tools that provide programming assignments to CS learners. However, the topic information of the problems provided in OJs is usually missing. Existing work on topic classification of CS problems is either coarse-grained or designed for advanced topics, and is not suitable for classifying the basic topics taught in a CS1 course. Thus, to address our RQ, we propose a predictive model that can detect topics of programming problems based on a CS1 syllabus. Our pipeline applies text embedding and augmentation techniques, along with several off-the-shelf classifiers (RF, SVM, CNN, RNN, etc) to perform the classification for seven topics. The best CNN model achieved an average F-1 score of 88.9%. In the following section we also discuss the applications and implications of topic classification for different stakeholders.

IX. APPLICATIONS AND IMPLICATIONS

In this study we propose and validate different combinations of ML and DL pipelines for problem topic prediction, based on the CS1 syllabus. We believe that the results of our best model are satisfactory and can be used for automatic annotation of topics in OJs. The automatic classification of topics can be applied in several teaching scenarios, from the formulation of exams, to even systems using automatic question recommendation, based on the student's skill level.

For students, annotation of topics would be useful because they would not have to carry out an exhaustive search for problems related to the CS1 topics in volumes of problems without any categorisation. Notice that this exhaustive search process can demotivate the student [12], [18], [30], [42]. Moreover, the ease of appropriateness for learners is vital to consider when developing a tool to facilitate learning. To illustrate (via a counterexample), the learner can access problems that are extremely incompatible with their level (e.g., graph problems for a CS1 student), which could discourage the programmer from using the OJ system.

Note that beyond maintaining student participation, OJ systems must take responsibility for providing a logical, cohesive structure of instruction for students, without disorganising their learning [1], [18], [19], [42]. For example, if a student is presented with a question beyond their level, they may attempt to research the solution on their own, resulting in the student gaining an incomplete grasp of this higher-level topic. This may discourage true learning in favour of copying or memorisation of the solution, leading to the development of poor programming habits. Such a disorganised process can also decelerate student learning, as the time the learner takes to study a subject beyond their level could have been

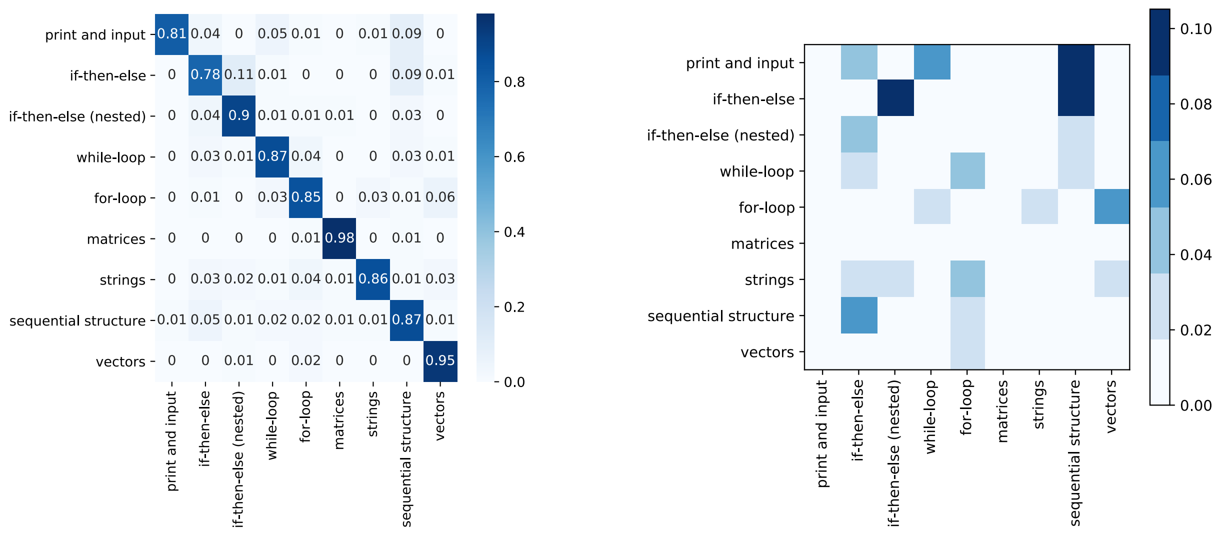


FIGURE 4. Confusion Matrix of our best model (left) and error density by topic (right). True values are presented on the rows whilst predicted values are depicted on the columns.

spent developing a strong understanding of material at an appropriate level.

For future work, our annotation is also useful to track what CS1 concepts the students are mastering based on the problems they solved and the ones they failed. Such works may verify the reasons why students fail a given problem of a specific topic provided by our model. This would be useful for creating interventions that provide meaningful feedback for students about the topics with which they struggle. Problems could also be recommended to strengthen the students' identified conceptual weaknesses.

For instructors, one of the most recurrent teaching tasks is the selection of questions to compose assignments and exams [6], [9], [12]. Assignments and exams are typically associated with a CS1 topic (e.g., an exam on the use of repetition structures) [12], [17]. With the automatic annotation of topics, instructors can access problems in an organised way, making their work easier and giving them more time to focus on other teaching tasks. This enables instructors to provide students with more coding examples, covering a wider breadth of scenarios and concepts than they would have access to otherwise due to the time constraints of filtering appropriate problems. When teaching programming principles, it is vital to cover the details of a certain topic in different contexts so that students develop an algorithmic understanding of how to approach similar problems. Without a large set of examples covering different aspects of the same topic, instructors are unable to sufficiently cover the various implementations possible, making it more difficult for students to generalise problem solutions from their limited experiences. Annotated problems alleviate the strain that instructors may face when searching for appropriate learning material.

Additionally, our method helps administrators of OJs, who could use our method to help organise problems in a way that is beneficial for students of different levels and instructors of different disciplines. While administrators of OJs could manually label future questions as they create them without much overhead, this can waste valuable unlabelled resources that already exist. Therefore, this model can not only improve future systems, but can also improve currently available systems.

Furthermore, note that our best model has an F1-score of 88.9%. In this way, classification errors may occur with the implementation of our model in a real scenario. In cases where there is a misclassification, the instructor, or even the students, could identify and correct it. The detection and correction process by instructors and even students could be relatively simple, since the CS1 topics are elementary. Before releasing an automatically generated exam, an instructor should check the generated problems (a relatively small task) and annotate those that are not appropriate (to be thus signalled to the algorithm), and ask for reallocation of those specific problems, until satisfied. Thus, the tendency is for errors to decrease with the use of the system, as humans relabel the problems.

Moreover, this human support would help the AI to improve its rules to and to become even more precise in annotating topics. Furthermore, such human/AI interaction is claimed by the literature [43], [44], [45] to be dominant in modern systems. The reason is that humans and AI have different strengths and weaknesses, and the combination of the heterogeneous intelligence of both agents can be quite powerful.

A. LIMITATIONS

When thinking about the task of classifying problems used in assignments and exams, our predictive model can hurt students by giving them a set of more complex or easier questions than desired. However, as our model has a high F1-score, classification errors would not occur frequently.

Furthermore, on a platform that combines human knowledge with AI, this limitation could be overcome by corrections (re-annotation) from students, instructors and OJ maintainers. That is, as a user encounters a misclassified problem, he/she could suggest the correct topic annotation.

Moreover, we define the problem tackled in this work as a multiclassification instead of multi-label classification. We did so because for the pedagogical MSP approach point of view, a multiclassification definition of the problem is more suitable since the students should solve the code questions and make progress gradually, achieving small single (in terms of topics) learning goals (e.g., mastering simple conditional structure, instead of mastering structural conditions – more general). Consequently, even if a problem could be associated with more the one topic, one of these topics would be more complex and dominate the other topic. In other words, the easier topic would be into the other topic (in terms of set theory), considering that content is taught in a cumulative way, as it generally is. To illustrate, consider a question of while topic that also needs conditional structures to be solved. In such a case, the while subject dominates the conditional structure concept since it is expected that a student masters conditional structure in order to learn the while structure in the MSP approach. Thus, there is no need to define the problem in a more difficult way (multi-label classification) without any benefits for instructors and students who follow the MSP approach. However, for other methodologies that employed in OJ systems that may use problems with multiple learning objectives, such multi-label definition of the problem might be more suitable for.

X. CONCLUSION, LIMITATIONS AND FUTURE WORK

CS1 classes have a high failure rate around the world, which is why it is so important to invest in tools that might support the teaching and learning process.

Our results provide proof that metrics extracted from the problems' descriptions can potentially be used to automate this process. Our approach can be applied to automatically and relatively accurately annotate problems in OJs with the CS1 curriculum. We also explain challenge, such as specific topics which increase the complexity of the solution. Moreover, although the results may not surpass human evaluators, our approach-allows for easy scalability, especially for large datasets, in which new questions are registered frequently. Notice that our method can quickly classify new questions, as the database of questions grows.

As future work, we intend to check the predictive power of our model in other online judges. Moreover, we intend to use our best model, coupled with a recommender system mechanism, to recommend problems for CS1 students and instructors.

ACKNOWLEDGMENT

The present work is the result of the Research and Development (R&D) project 001/2020, signed with Federal University of Amazonas and FAEPI, Brazil, which has funding from Samsung, using resources from the Informatics Law for the Western Amazon (Federal Law n° 8.387/1991), and its disclosure is in accordance with article 39 of Decree No. 10.521/2020. This study was financed in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil - CNPq (Process 308513/2020-7), Fundação de Amparo à Pesquisa do Estado do Amazonas - FAPEAM (Process 01.02.016301.02770/2021-63), and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Moreover, this study was financed in part by the Acuity Insights under the Alo Grant program.

REFERENCES

[1] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A survey on online judge systems and their applications," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 1–34, Jan. 2019.

[2] C. M. Intisar, Y. Watanobe, M. Poudel, and S. Bhalla, "Classification of programming problems based on topic modeling," in *Proc. 7th Int. Conf. Inf. Educ. Technol.*, Mar. 2019, pp. 275–283.

[3] F. D. Pereira, E. Oliveira, A. Cristea, D. Fernandes, L. Silva, G. Aguiar, A. Alamri, and M. Alshehri, "Early dropout prediction for programming courses supported by online judges," in *Proc. Int. Conf. Artif. Intell. Educ.* Cham, Switzerland: Springer, 2019, pp. 67–72.

[4] A. A. Sánchez-Ruiz, G. Jimenez-Diaz, P. P. Gómez-Martín, and M. A. Gómez-Martín, "Case-based recommendation for online judges using learning itineraries," in *Proc. Int. Conf. Case-Based Reasoning*. Cham, Switzerland: Springer, 2017, pp. 315–329.

[5] D. Joyner, "Toward CS1 at scale: Building and testing a MOOC-for-credit candidate," in *Proc. 5th Annu. ACM Conf. Learn. Scale*, Jun. 2018, pp. 1–10.

[6] C. L. Gordon, R. Lysecky, and F. Vahid, "The rise of program auto-grading in introductory CS courses: A case study of zyLabs," in *Proc. ASEE Virtual Annu. Conf. Content Access*, 2021.

[7] F. D. Pereira, L. M. de Souza, E. H. T. de Oliveira, D. B. F. de Oliveira, and S. G. L. de Carvalho, "Predição de desempenho em ambientes computacionais para turmas de programação: Um mapeamento sistemático da literatura," in *Proc. Anais 31st Simpósio Brasileiro Informática Educação*, 2020, pp. 1673–1682.

[8] H. Sun, B. Li, and M. Jiao, "YOJ: An online judge system designed for programming courses," in *Proc. 9th Int. Conf. Comput. Sci. Educ.*, Aug. 2014, pp. 812–816.

[9] J. L. Bez, N. A. Tonin, and P. R. Rodegheri, "URI online judge academic: A tool for algorithms and programming classes," in *Proc. 9th Int. Conf. Comput. Sci. Educ.*, Aug. 2014, pp. 149–152.

[10] A. G. de Oliveira Fassbinder, T. G. Botelho, R. J. Martins, and E. F. Barbosa, "Applying flipped classroom and problem-based learning in a CS1 course," in *Proc. IEEE Frontiers Educ. Conf.*, Oct. 2015, pp. 1–7.

[11] C. Hogg and M. Jump, "Designing autograders for novice programmers," in *Proc. 53rd ACM Tech. Symp. Comput. Sci. Educ.*, Mar. 2022, p. 1200.

[12] F. D. Pereira, H. B. F. Junior, L. Rodriguez, A. Toda, E. H. T. Oliveira, A. I. Cristea, and D. B. F. Oliveira, "A recommender system based on effort: Towards minimising negative affects and maximising achievement in CS1 learning," in *Proc. Int. Conf. Intell. Tutoring Syst.* Cham, Switzerland: Springer, 2021, pp. 466–480.

[13] J. Allen, F. Vahid, K. Downey, and A. Edgcomb, "Weekly programs in a CS1 class: Experiences with auto-graded many-small programs (MSP)," in *Proc. ASEE Annu. Conf. Expo.*, 2018.

[14] J. M. Allen and F. Vahid, "Concise graphical representations of Student effort on weekly many small programs," in *Proc. 52nd ACM Tech. Symp. Comput. Sci. Educ.*, Mar. 2021, pp. 349–354.

[15] F. D. Pereira, E. H. T. Oliveira, D. B. F. Oliveira, A. I. Cristea, L. S. G. Carvalho, S. C. Fonseca, A. Toda, and S. Isotani, "Using learning analytics in the Amazonas: Understanding students' behaviour in introductory programming," *Brit. J. Educ. Technol.*, vol. 51, no. 4, pp. 955–972, Jul. 2020.

[16] I. Karvelas and B. A. Becker, "Sympathy for the (novice) developer: Programming activity when compilation mechanism varies," in *Proc. 53rd ACM Tech. Symp. Comput. Sci. Educ.*, Feb. 2022, pp. 962–968.

[17] J. M. Allen, F. Vahid, A. Edgcomb, K. Downey, and K. Miller, "An analysis of using many small programs in CS1," in *Proc. 50th ACM Tech. Symp. Comput. Sci. Educ.*, Feb. 2019, pp. 585–591.

[18] W. X. Zhao, W. Zhang, Y. He, X. Xie, and J.-R. Wen, "Automatically learning topics and difficulty levels of problems in online judge systems," *ACM Trans. Inf. Syst.*, vol. 36, no. 3, pp. 1–33, Jul. 2018.

[19] F. D. Pereira, F. Pires, S. C. Fonseca, E. H. T. Oliveira, L. S. G. Carvalho, D. B. F. Oliveira, and A. I. Cristea, "Towards a human-AI hybrid system for categorising programming problems," in *Proc. 52nd ACM Tech. Symp. Comput. Sci. Educ.*, Mar. 2021, pp. 94–100.

[20] S. C. Fonseca, F. D. Pereira, E. H. Oliveira, D. B. Oliveira, L. S. Carvalho, and A. I. Cristea, "Automatic subject-based contextualisation of programming assignment lists," in *Int. Educ. Data Mining Soc.*, 2020.

[21] L. N. Gamage, "A bottom-up approach for computer programming education," *J. Comput. Sci. Colleges*, vol. 36, no. 7, pp. 66–75, 2021.

[22] L. Rodrigues, F. Pereira, A. Toda, P. Palomino, W. Oliveira, M. Pessoa, L. Carvalho, D. Oliveira, E. Oliveira, A. Cristea, and S. Isotani, "Are they learning or playing? Moderator conditions of Gamification's success in programming classrooms," *ACM Trans. Comput. Educ.*, vol. 22, no. 3, pp. 1–27, Sep. 2022.

[23] A. Godea, F. Bulgarov, and R. Nielsen, "Automatic generation and classification of minimal meaningful propositions in educational systems," in *Proc. COLING*, 2016, pp. 3226–3236.

[24] X. Wang, S. T. Talluri, C. Rose, and K. Koedinger, "UpGrade: Sourcing student open-ended solutions to create scalable learning opportunities," in *Proc. 6th ACM Conf. Learn. Scale*, Jun. 2019, pp. 1–10.

[25] C. Charitsis, C. Piech, and J. C. Mitchell, "Using NLP to quantify program decomposition in CS1," in *Proc. 9th ACM Conf. Learn. Scale*, Jun. 2022, pp. 113–120.

[26] Q. Liu, Z. Huang, Y. Yin, E. Chen, H. Xiong, Y. Su, and G. Hu, "EKT: Exercise-aware knowledge tracing for student performance prediction," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 1, pp. 100–115, Jan. 2021.

[27] A. Géron, *Hands-on Machine Learning With Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[28] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," 2021, *arXiv:2107.13586*.

[29] V. Athavale, A. Naik, R. Vanjape, and M. Shrivastava, "Predicting algorithm classes for programming word problems," in *Proc. 5th Workshop Noisy User-Generated Text (W-NUT)*, Hong Kong, 2019, pp. 84–93.

[30] H. B. D. F. Junior, F. D. Pereira, E. H. T. D. Oliveira, D. B. F. D. Oliveira, and L. S. G. D. Carvalho, "Recomendação automática de problemas em Juízes online usando processamento de linguagem natural e análise dirigida aos dados," in *Proc. Brazilian Symp. Comput. Educ.*, Nov. 2020.

[31] T. Aljohani, F. D. Pereira, A. I. Cristea, and E. Oliveira, "Prediction of users' professional profile in MOOCs only by utilising learners' written texts," in *Proc. Int. Conf. Intell. Tutoring Syst.* Cham, Switzerland: Springer, 2020, pp. 163–173.

[32] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.

[33] E. Ma. (2019). *NLP Augmentation*. [Online]. Available: https://github.com/makcedward/nlpaug

[34] M. Raghu and E. Schmidt, "A survey of deep learning for scientific discovery," 2020, *arXiv:2003.11755*.

[35] S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems*. Sebastopol, CA, USA: O'Reilly Media, 2020.

[36] A. Bartoli and A. Fusiello, *Computer Vision—ECCV 2020 Workshops*. Glasgow, U.K.: Springer, Aug. 2021, vol. 12540.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, and M. Blondel, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[38] W. L. Hamilton, K. Clark, J. Leskovec, and D. Jurafsky, "Inducing domain-specific sentiment lexicons from unlabeled corpora," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2016, p. 595.

[39] W. L. Hamilton, J. Leskovec, and D. Jurafsky, "Diachronic word embeddings reveal statistical laws of semantic change," 2016, *arXiv:1605.09096*.

[40] M. Malekzadeh, P. Hajibabaee, M. Heidari, S. Zad, O. Uzuner, and J. H. Jones, "Review of graph neural network in text classification," in *Proc. IEEE 12th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Dec. 2021, pp. 0084–0091.

[41] B. Gupta, P. Prakasam, and T. Velmurugan, "Integrated BERT embeddings, BiLSTM-BiGRU and 1-D CNN model for binary sentiment classification analysis of movie reviews," *Multimedia Tools Appl.*, vol. 81, no. 23, pp. 1–20, 2022.

[42] R. Yera and L. Martínez, "A recommendation approach for programming online judges supported by data preprocessing techniques," *Appl. Intell.*, vol. 47, pp. 277–290, Mar. 2017.

[43] D. Dellermann, P. Ebel, M. Söllner, and J. M. Leimeister, "Hybrid intelligence," *Bus. Inf. Syst. Eng.*, vol. 61, no. 5, pp. 637–643, 2019.

[44] K. Holstein, V. Aleven, and N. Rummel, "A conceptual framework for human–AI hybrid adaptivity in education," in *Proc. Int. Conf. Artif. Intell. Educ.* Cham, Switzerland: Springer, 2020, pp. 240–254.

[45] F. D. Pereira, S. C. Fonseca, E. H. T. Oliveira, A. I. Cristea, H. Bellhauser, L. Rodrigues, D. B. F. Oliveira, S. Isotani, and L. S. G. Carvalho, "Explaining individual and collective programming students' behavior by interpreting a black-box predictive model," *IEEE Access*, vol. 9, pp. 117097–117119, 2021.

FILIPE DWAN PEREIRA received the B.S. degree in computer science from the Federal University of Roraima and the M.S. degree in computer science from the Federal University of Amazonas, where he is currently pursuing the Ph.D. degree in artificial intelligence applied to education. Since 2013, he has been a Auxiliar Professor with the Department of Computer Science, Federal University of Roraima. His research interests include education data mining, learning analytics, artificial intelligence, machine learning, big data, computing in education, and information systems.

SAMUEL C. FONSECA is currently pursuing the bachelor's degree in computer engineering with the Federal University of Amazonas. He also works as a Software Developer with SIDIA, the largest research and development institute in Latin America. His research interests include machine learning, computer vision, and natural language processing.

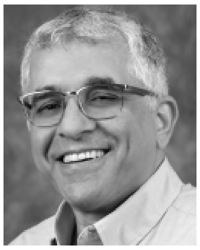
SANDRA WIKTOR is currently pursuing the Ph.D. degree with the University of North Carolina, Charlotte, focusing on computer science education and learning analytics using natural language processing and machine learning techniques. Her research interests include text-based emotion detection with reflective writing, investigating student sentiment toward a learning environment, and intervention methods.

DAVID B. F. OLIVEIRA received the Doctoral degree in computer science from the Federal University of Minas Gerais, in 2010. He is currently a Professor with the Institute of Computing, Federal University of Amazonas, Brazil, where he works as a Researcher, a Professor, and an Advisor in undergraduate. He has experience in information retrieval and informatics in education areas. He is also the Development Team Manager of the CodeBench System, which is an online judge, which automatically grades the programming assignments submitted by students. He has worked on the following research topics: search for structured content, online judges, gamification of educational systems, and web development.

ALEXANDRA I. CRISTEA (Senior Member, IEEE) is a Professor, the Head of the Artificial Intelligence and Human Systems (AIHS) Group, and the Deputy Head with the Computer Science Department, Durham University; and an Honorary Professor with the Computer Science Department, Warwick University. Her research interests include web science, learning analytics, user modeling and personalization, semantic web, and social web. She has authored over 300 papers on these subjects (over 4000 citations on Google Scholar, H-index is 35). She was classified within the top 50 researchers in the world in the area of educational computer-based research according to Microsoft Research. She has been highly active and has an influential role in international research projects. She has led various projects, has been keynote/invited speaker, an organizer, a co-organizer, a panelist, and a program committee member of various conferences in her research field. She is a member of the Editorial Board of the IEEE Transactions on Learning Technologies, an Executive Peer-Reviewer of the IEEE LTTF Education Technology and Society Journal, and an Associate Editor of *Frontiers in Artificial Intelligence*.

AILEEN BENEDICT is currently pursuing the Ph.D. degree with the University of North Carolina, Charlotte. She is also a GAANN Fellow with the University of North Carolina, who has been mentored in teaching, since 2016. Her work mainly focuses on computer science education and learning analytics, with specific interests in reflective practices and predictive analytics. She also has research interests in recommender systems and human-centered AI.

MOHAMMADALI FALLAHIAN is currently pursuing the Ph.D. degree in computer science with the University of North Carolina, Charlotte. He has been working as a Senior Software Developer for over ten years. His research interests include deep learning, generative models, big data, and approximate query processing. In addition, he has collaborated actively with researchers in designing software patterns and architecture on applied machine learning projects.



MOHSEN DORODCHI is a Full Teaching Professor of computer science with the University of North Carolina, Charlotte. His research interests include learning and predictive analytics/visualization, teaching innovation, computer science education research, software engineering, and broadening participation in computing. His research has been supported by the National Science Foundation (NSF) and state and local organizations and industries.

LEANDRO S. G. CARVALHO received the B.S. degree in electronic engineering from the Technological Institute of Aeronautics, Brazil, in 2000, and the M.S. and D.S. degrees in informatics from the Federal University of Amazonas, Brazil, in 2004 and 2011, respectively. Since 2006, he has been a Professor with the Institute of Computing, Federal University of Amazonas. His research interests include computer science education and educational data mining. He has been an organizer, a co-organizer, and a program committee member of various conferences in his research field.

RAFAEL FERREIRA MELLO holds a permanent faculty position with the Federal Rural University of Pernambuco, Recife, Brazil, where he is one of the coordinators of the AIBox Laboratory. He has worked on several multinational research projects involving institutional and organizational partners in Europe, Australia, and Latin America. He has coauthored 84 publications, including one book, four book chapters, 18 journal articles, and 61 refereed conference papers. A key theme of his recent research has been the use of natural language processing in applied fields, such as education and for analyzing the content of a range of documents through text-summarization and topic-modeling algorithms.

ELAINE H. T. OLIVEIRA received the Diploma degree in computer science from the University of São Paulo, Brazil, and the Ph.D. degree from the Graduate Program in Informatics in Education, Federal University of Rio Grande do Sul, Brazil, in 2011. Since 2002, she has been a Professor with the Institute of Computing, Federal University of Amazonas. Her current research is focused on studying students' behavior as them learn how to program, using learning paths and data-driven approaches. The data is collected by the interaction of the students with a self-devised online judge and learning management systems. The goals of her research are to predict outcomes, help decision making, and provide adaptive learning through learning analytics. She is an Associate Editor of the *Brazilian Journal of Computers in Education* (2019–2021).