Automatic Subject-based Contextualisation of Programming Assignment Lists

Samuel C. Fonseca
Institute of Computing
Federal University of Amazonas
Manaus, Brazil
scf@icomp.ufam.edu.br

Filipe Dwan Pereira
Department of Computer Science
Federal University of Roraima
Boa Vista, Brazil
filipe.dwan@ufrr.br

Elaine H. T. Oliveira
Institute of Computing
Federal University of Amazonas
Manaus, Brazil
elaine@icomp.ufam.edu.br

David B. F. Oliveira
Institute of Computing
Federal University of Amazonas
Manaus, Brazil
david@icomp.ufam.edu.br

Leandro S. G. Carvalho
Institute of Computing
Federal University of Amazonas
Manaus, Brazil
galvao@icomp.ufam.edu.br

Alexandra I. Cristea
Department of Computer Science
Durham University
Durham, United Kingdom
alexandra.i.cristea@durham.ac.uk

ABSTRACT

As programming must be learned by doing, introductory programming course learners need to solve many problems, e.g., on systems such as 'Online Judges'. However, as such courses are often compulsory for non-Computer Science (non-CS) undergraduates, this may cause difficulties to learners that do not have the typical intrinsic motivation for programming as CS students do. In this sense, contextualised assignment lists, with programming problems related to the students' major, could enhance engagement in the learning process. Thus, students would solve programming problems related to their academic context, improving their comprehension of the applicability and importance of programming. Nonetheless, preparing these contextually personalised programming assignments for classes for different courses is really laborious and would increase considerably the instructors'/monitors' workload. Thus, this work aims, for the first time, to the best of our knowledge, to *automatically classify the programming assignments in Online Judges based on students' academic contexts* by proposing a *new context taxonomy*, as well as a *comprehensive pipeline evaluation methodology* of cutting edge competitive Natural Language Processing (NLP). Our comprehensive methodology pipeline allows for comparing state of the art data augmentation, classifiers, beside NLP approaches. The context taxonomy created contains 23 subject matters related to the non-CS majors, representing thus a challenging multi-classification problem. We show how even on this problem, our comprehensive pipeline evaluation methodology allows us to achieve an accuracy of 95.2%, which makes it possible to automatically create contextually personalised program assignments for non-CS with a minimal error rate (4.8%).

Keywords

non-CS majors, NLP, contextually personalised assignment lists

1. INTRODUCTION

Introductory Programming (often known under the label of 'CS1') classes are now-a-days often compulsory for undergraduate courses that do not have computing as their major [10, 15, 20, 23]. CS1 is delivered to students majoring in, e.g., mechanical engineering, economics, etc. - whom we collectively name here 'non-CS students'. It is common in such cases to find students with difficulty in interpreting assignment texts, due to the lack of affinity with the area of the problem [22]. As a result, many of these students may be discouraged by CS1, as they fail to see the purpose that programming can have in their professional lives [10, 17, 23].

Moreover, programming must be learned by doing and, hence, learners need to solve many problems [11, 17–19, 27]. In this sense, 'Online Judge' systems can influence positively the learning process of non-CS students [12, 18, 20, 25], as systems which allow students to submit programming assignments and provide real-time automatic code correction. As Programming Online Judges (POJ) have large numbers of problems registered in their problem banks [25], in principle, there would be plenty of problems to select from, for both students as well as teachers, allowing for a mass personalisation - where one teacher could cater in parallel for the needs of many students. Nonetheless, the problems available on these systems often are collected or scraped from various environments that do not provide labelling [27], and thus it is laborious to find appropriate problems for non-CS students. This is more so the case, as the number of programming exercises is constantly increasing [25, 27]. Therefore,

Samuel Fonseca, Filipe Dwan Pereira, Elaine H. T. Oliveira, David Fernandes, Leandro Carvalho and Alexandra Cristea "Automatic Subject-based Contextualisation of Programming Assignment Lists" In: *Proceedings of The 13th International Conference on Educational Data Mining (EDM 2020)*, Anna N. Rafferty, Jacob Whitehill, Violetta Cavalli-Sforza, and Cristobal Romero (eds.) 2020, pp. 81 - 91

the automatisation of the categorisation of problems based on subject matter is becoming vital, to support instructors who teach computer programming disciplines. To illustrate, undergraduate students of Economics would be more familiar with an *if-then-else* problem using terms such as "interest rates" or "importation of goods" instead of a problem on the "growth of cells", which may be completely out of their comfort zone. Thus, we raise the following research question:

How can we extract the subject matter from programming problem statements, to automatically match programming assignment lists to non-CS courses?

Our main contributions with this paper are thus:

- Proposing a new, *wholistic methodology pipeline for the POJ contextual labelling problem*, allowing to compare a variety of cutting edge shallow and deep learning models, to experiment with the most recent data augmentation techniques (with or without augmentation), NLP (based on BERT, Word2Vec, Glove), classifiers (based on BERT, Random Forest, SVM, XGBoost, GaussianNB, GradientBoosting, ExtraTree, Sequential DNN, CNN, RNN) and validation.

- Extracting, for the first time, to the best of our knowledge, automatically and precisely, subject matters related to non-CS courses; we do this by using cutting edge NLP techniques on the statements of assignments available in a home-made online judge CodeBench[1] used with fifteen non-CS major programmes.

- Proposing a *subject-based contextualisation taxonomy* to map subject matters to non-CS courses, where CS1 is compulsory.

- We thus are enabling the contextual personalisation of programming assignment lists for non-CS courses.

2. RELATED WORK

There are many studies tackling the challenge of teaching introductory programming to non-CS students, based on a variety of angles. To illustrate, [10] employed collaborative scenarios to enhance teaching and learning programming in non-CS courses, whist [23] used an approach involving games and media. [15, 24] show that English-like (natural language) syntax can help non-CS students overcome the difficulties in learning programming syntax. Furthermore, a recent study [21] explains that effective motivational educational design can enhance introductory programming students and teacher engagement. Despite these works representing a move towards improving non-CS students engagement, linking text collections to general or domain-specific knowledge is essential [1,5]. More specifically, [14] argue that students' experiences of the learning context have important implications for teaching and learning. Nevertheless, none of these aforementioned studies take the context of the problem into account. Especially untouched is the issue of contextualisation of the problem statements, ensuring that problems introduce only the degree of difficulty required to progress in the programming knowledge and not additional complexity from strange contexts for the current learner (such as a geology context for economy students, etc.).

Online judges (POJ) are increasingly being used to support introductory programming (CS1) classes. Via such environments, teachers can provide problems to be solved and students can submit their code and receive immediate feedback [9, 18, 25]. One of the issues of these systems is that, in general, the problems available are not categorised based on subject matter, topics, context, major, etc. In this sense, there are two recent works [3, 27] which tackle the problem of topic extraction from such problems. In these studies, topic extraction is used for grouping problems in terms of their related programming knowledge components, concepts or skills. For example, a problem that can be solved by using graph algorithms, such as breadth-first search, flood-fill or topological sort, can be classified into the graph category. Notice however that the target audience of these studies are more experienced POJ users. Instead, here we are not interested in categorising problems based on advanced topics.

In fact, we tackle, for the first time, to the best of our knowledge, the challenge of extracting the subject matter from programming problem statements available in POJ systems used in introductory programming, in order to improve the teaching and learning process of CS1 for non-CS courses, by matching problems to non-CS majors.

3. EDUCATIONAL CONTEXT

In this paper, we use as study base, as said, the CodeBench Online Judge environment, which is self-designed and implemented, as it allows us the freedom to add the changes inspired by our research results. Thus, we analyse here running the Introductory Programming (CS1) course at the Federal University of the Amazonas, via this self-designed POJ, which is delivered to 15 non-CS undergraduate degrees across the university. These courses are divided into 5 major areas: Mathematics, Physics, Engineering, Statistic and Geology. Three of the degrees belong to Mathematics, 2 to Physics, 8 to Engineering, 1 to Statistics and 1 to Geology. Figure 1 illustrates this configuration.

As Figure 2 illustrates, during the CS1 course, students in our environment typically solve 7 assignment lists with problems of increasing difficulty, using the Python programming language. They are allowed to solve the problems with an unlimited number of submission attempts, as long as they meet the deadline for solving all problems on a given list. The exercise lists always precede an exam on the same programming topic, both carried out in the Online Judge. Each list has an average of 10 questions, and the tests have 2 questions. We call a list together with its exam a 'session', where each session addresses a specific programming topic. Altogether, the course thus is formed of 7 sessions, that is, 7 programming topics are covered during CS1. Each session lasts on average 2 weeks.

During the 7 sessions, students work on the following programming topics: *Sequential*, *Composite conditional structures*, *Chained conditional structures*, *Repeating structures by condition*, *Repeating structures by count*, *Vectors and Strings* and *Matrices*. Before the 7 sessions, students have a

[1]http://codebench.icomp.ufam.edu.br/index.php

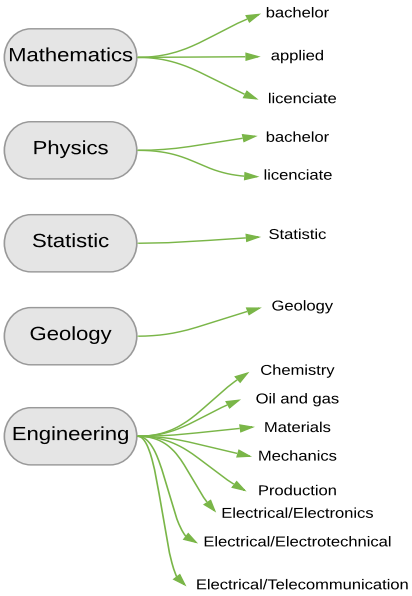

Figure 1: non-CS undergraduate courses at the Federal University of the Amazonas

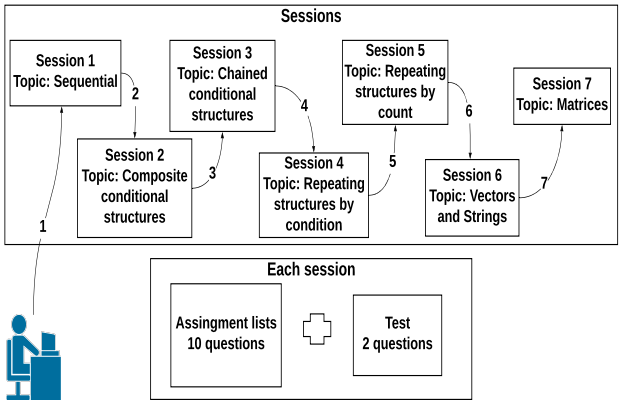

Figure 2: CS1 course configuration

first week to get used to the Python programming language, where they learn about *Variables* and *Single Operations*.

Whilst, in our online judge, problems are well structured based on these programming topics as above, they lack a clear division based on the contexts (here, related major areas) in which the problems are to be delivered. Please also note that, although the sessions are ordered by their increasing difficulty, the topics they are addressing are somewhat unrelated. Moreover, this increase in difficulty is typical for any CS1 course, be it offline or online.

Thus, our POJ is generic enough and is hence a good environment in which to research approaches to automatic classification by contexts, based on the statements, to build context-based personalised assignment lists, towards ultimately enhancing the engagement of non-CS students in their learning process.

4. DATA

The database in our Online Judge system consists of 986 programming problems in the CS1 discipline. As said, the statements in the database were initially not categorised by context; thus, we proceeded to create a labelled corpus, by manually classifying the contexts of each statement, to further use to carry out the experiments.

As labels, we adopted in this research contexts extracted from Zanini and Raabe's definitions [26], which show that the context of problems plays an important role for novice programming students. Their study manually analysed the contexts of 428 programming problems statements used in introductory programming (as in our case) offered to 51 undergraduate courses. As a result, they found 20 possible contexts for these problems, as follows: mathematical, commercial, person, school, human resources, research, banking, physics, production, sport, computational, traffic, date and time, environment, tax, safety, consumption, population, others, and gamble.

We thus started with their proposed labels to annotate our problems. However, there were some groups of statements that could not be mapped over the above contexts. Moreover, the context "others" is too general and provides no real information. Given that, we removed the context "others" and propose here some additional contexts, as part of our contribution, in order to annotate our larger set of statements. As a result of the above process, we produced a total of 23 contexts, which we grouped together in a *new CS1 Context Taxonomy*, which is described in Table 1. This includes the following contexts, as contributions of our research: *Games*, *Movies and Series*, *Chemistry* and *Geography*. In addition, the table shows the number of statements for each context labelled and used in this research, the description of the contexts as well as the undergraduate courses that may have a high connection with the context.

It is worth noting that we performed a statistic test that measures inter-annotator agreement to validate if our annotation process was conducted properly. To do so, we used Cohen's kappa (k) [4], which shows the level of agreement between two annotators on a classification task. As a result, we achieved a $k = 0.961$, which is considered almost perfect agreement [2].

5. METHODOLOGY

Figure 3 illustrates the proposed evaluation methodology pipeline used in the experiments of our research. We create here a unique, comprehensive pipeline, studying various combinations of the most popular and successful bleeding edge state-of-the-art techniques for natural language processing (NLP). The following subsections explain each step of our methodology.

5.1 Data augmentation

The data augmentation stage consists of balancing the training data by paraphrasing it, using the pre-trained model BERT [6]. Importantly for our task, this allows for *contex-*

Context	Focus of the Statement	non-CS course	N
Mathematical	resolution of purely mathematical problems, without this being applied to another context	Mathematics and Engineering	261
Commercial	handling of products, goods, such as buying and selling, calculation of commission, provision of services	Economy	120
Games	game application, be it a virtual game or even a table game; for example, in the database there are games of naval battles, as well as video games	Digital games courses	96
School	to solve a school problem, such as averaging, passing or failing verification	Pedagogy	79
Traffic	related to the driver, car, mileage, accidents	All courses	43
Sport	some activity involved with sport, such as running, football, classification	Physical education	42
Physics	resolution of purely physical problems, without this being applied to another context	Physics and Engineering	36
Banking	related to bank transactions, investment, balance, withdrawal, deposit, stock exchange	Economy	35
Human Resources	problem related to human resources, such as salary calculation, data related to employees, calculation of bonuses, recruitment and selection of employees	Sociology and Psychology	35
Movies and TV Shows	problem situation in a film or TV shows. To illustrate, there are questions from the movie *Harry Potter* about potion calculation	All courses	30
Population	problems on population data, such as birth rate, mortality rate, population growth; referring to either human or animal population	Statistic	25
Chemistry	purely chemical problems, without this being applied to another context	Chemical engineering	23
Person	problems with elements directly related to a person, like weight, height, sex	All courses	22
Date and time	calculation of date or time, calculation of day, verification of month, conversion of hours, minutes and seconds, time interval	All courses	21
Safety	control access, password verification, data security, encryption, validation	Software engineering	20
Research	providing statistical data of opinion polls	Statistic and Journalism	18
Environment	relating to environmental issues, such as pollution, temperature	Environmental engineering	18
Health	related to issues of fighting diseases	Medicine	17
Consumption	calculation of water, electricity or telephone-related consumption	Economy	16
Geography	resolution of purely geographical problems, without this being applied to another context	Geology	11
Production	related to the production of products, the quantity produced, production value, origin of the products	Production engineering	7
Computational	computational issues, such as conversion of binary, decimal, hexadecimal numbers, ASCII table	Computer engineering	6
Tax	calculation of taxes, such as income tax	Economy	5

Table 1: Our proposed CS1 Context Taxonomy and Data Set description, with respective non-CS undergraduate course name and Number of items per Context, N

tual paraphrasing. Figure 4 illustrates a paraphrasing process based on a fragment of a statement from the category "Computational".

Figure 4 shows a new generated sentence with clear semantics for a human reader. Still, generated text sometimes misses such a clear structure. Nevertheless, our goal here is not to generate new sentences which could be meaningful for learners. Instead, we aim at creating artificial statements, which are not to be presented to humans, but will be used to expand the minority classes, providing variations to the predictive models (see bias-variance trade-off [8]). In other

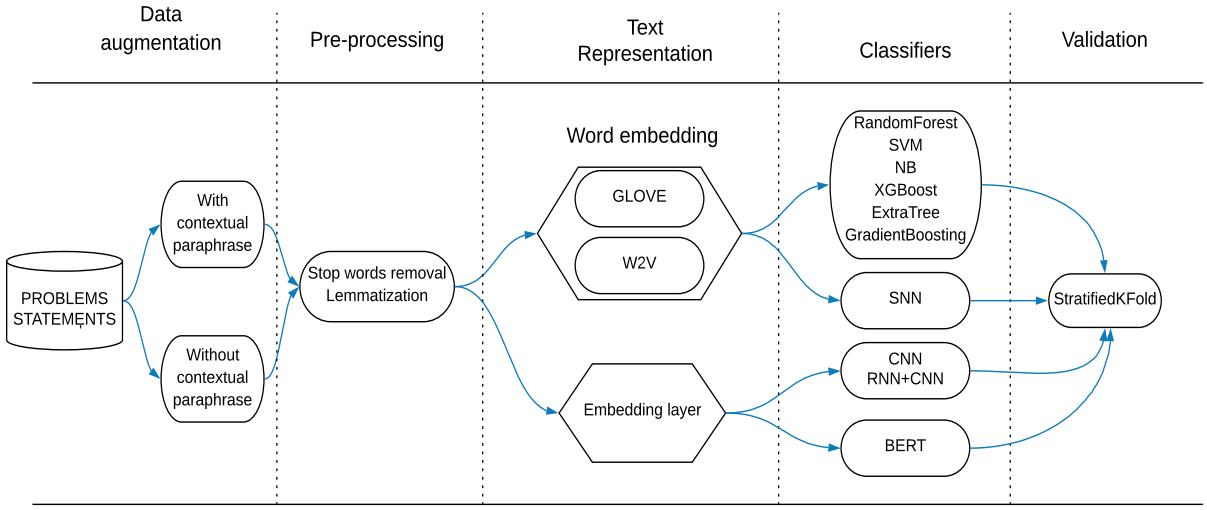

Figure 3: Proposed Automatic Contextualisation Research Methodology Pipeline

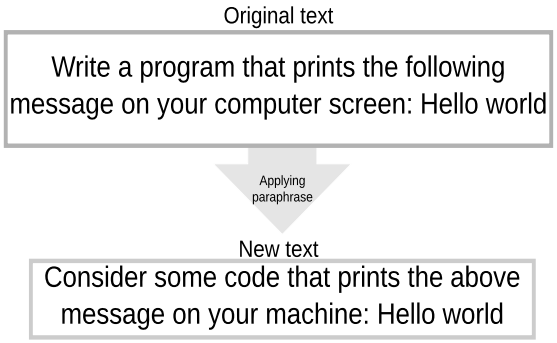

Figure 4: Paraphrasing Example using BERT [6]

words, despite this irregularity in the semantic sense of the statement, it is possible to perceive that the new instance generated belongs to the same class from which it was derived from and, therefore, it may represent a useful addition for the learning algorithm (which is later, as can be seen, confirmed by the results).

Nonetheless, as can be seen, despite the potential of such contextual paraphrasing, the new statements repeat some words from the original and keep almost the same number of tokens, which is a limitation of this method. As such, to prevent overtraining on artificial data (instances created using contextual paraphrases), we have set a limit of, at most, quadrupling the base of minority classes. We established this limit after some empirical experiments. That is, a statement is allowed to generate at most 4 new samples in the training base, as long as the new number of statements is below the number of instances of the majority class. Hence, this process may not render a perfectly balanced training base. To illustrate, imagine that the majority class has 10 questions on the training set, while the minority class has 1 question; with this paraphrasing algorithm, it is possible to extend the minority class for up to 5 questions (4 new samples + original statement).

In this work, experiments were carried out with and without paraphrasing, in order to analyse how the balancing by paraphrasing can influence the results.

5.2 Pre-processing

As we used reliable data (problems statements created directly by instructors/monitors), there was no need in our data processing of performing orthographic corrections, expanding contractions and other common data-cleaning steps. However, all our problem statements were originally in the Portuguese language. As there are many tools available for processing text written in English, we opted to translate our statements first into English, by using the *googleTrans*[2] library. Subsequently, we proceeded in applying our pipeline processing on the English text obtained, with and without the use of *stop-words* removal and *lemmatisation*, using spacy[3]. As a result, we observed empirically that these two techniques were useful for data filtering in our pre-processing step. Next, we show how we further prepare the text for the machine learning algorithms.

5.3 Text Representation

The machine learning algorithms take as input a sequence of text to learn the structure of text, just like a human does. However, we need to convert the data in numerical form. As such, we represent our text data as a sequence of numbers (see Keras Tokenizer function[4]). Moreover, the ML algorithm expects each training instance to have the same length (same number of tokens). Thus we padded with zeros at the end sequences that are shorter than the maximum length

[2]pypi.org/project/googletrans/
[3]spacy.io
[4]keras.io/preprocessing/text/

sequence. To do so, we applied the Keras padding module[5] over the sequences.

In addition, two different state-of-the-art NLP techniques for vector representation of words are used for competing against each other: googleNews-Vectors (W2V)[6] and Glove [16] word embeddings. Moreover, for the BERT classifier, we used its own layer of word embeddings. Similarly, for the other deep learning models, we used the word embeddings layers as provided by the Keras library[7]. The purpose of this step is to compare the NLP techniques in terms of performance with our data set. Therefore, we created a *process to obtain the best model for automatic categorisation of contexts of programming questions* for our educational context. The process allowed us thus to carry out experiments with advanced Deep Learning methods, and to compare not only those approaches with each other, but also with classical approaches, such as shallow learning models.

5.4 Classifiers

For deep learning models we used: a) Convolutional Neural Networks (CNN) which have a convolutional layer, followed by three dense layers; b) Recurrent Neural Networks (RNN), with a recurring layer using a Long Term and Short term memory (LSTM) followed by three dense layers; c) RNN and CNN (RNN+CNN) stacked with the same configurations as those of the items a and b; d) Sequential Neural Network (SNN) with two dense layers and e) BERT for classification (notice that we used BERT for two purposes: i) perform contextual paraphrasing; ii) multi-classification).

As we are tackling a multi-classification problem, the final layer for each neural network was represented by a softmax layer [13]. For all deep learning models, the configurations used above represent the default recommended ones from the literature [13].

Additionally, we used the following classical, shallow classifiers, with the word embeddings from googleNews-Vectors and Glove: Random Forest Classifier (RFC), Support Vector Machine (SVM), Extremely Randomised Tree Classifier (ETC), Gaussian Naive Bayes (GNB), XGBoost (XGB) and Gradient Boosting Classifier (GBC).

5.5 Validation

To validate the models, we employed the stratified validation with 10 *folds*. This method divides the base into k partitions, using $k-1$ for training and 1 for testing. After that, the accuracy of the test partition is calculated. This process is repeated k times, until all partitions have been used as a test. Finally, the average of the accuracy obtained in the tests is computed. It is noteworthy that each *fold* was divided proportionally to the number of statements present in each class in the database [13]. We implemented it using the *StratifiedKFold* from scikit-learn. Notice that we performed the data augmentation only on the training sets of each training fold. Thus, there were no paraphrased texts in the test sets.

[5]keras.io/preprocessing/sequence/
[6]code.google.com/archive/p/word2vec/
[7]keras.io/

To evaluate our models, we used the F1-score, as this metric combines precision and recall in an harmonic mean. This is useful because it gives much more weight to low values than a regular mean, which treats all values equally. Moreover, we used the weighted F1-score, which takes into account the proportion of each class.

6. RESULTS AND DISCUSSION

We built a total of 34 predictive models. Figure 5 illustrates all the results obtained by all models applied in this research. From this figure, we can notice that paraphrasing improved the (weighted) *F1-score* in all models. To illustrate this boosting, the model *GLOVE + SVM* achieved a F1-score of 86%, without paraphrasing. Whereas with the paraphrasing, the model achieved 94%, an increase of 8%. To validate that, we performed the McNemar's hypothesis statistical test, which is recommended to compare machine learning models [7]. We compared the models with or without the contextual paraphrasing. As a result, we confirmed that the paraphrasing statistically boosts all models, even after Bonferroni correction ($p - values \ll 0.05/2$). Table 2 shows the classification performance of the models in terms of macro and weighted precision, recall and F1-score. Moreover, this table shows the accuracy of each model.

From a visual inspection of Figure 5, we can argue that the best model found is the BERT classifier with use of the contextual paraphrasing (BERT + PAR), as the model has the highest median and a low standard deviation. Moreover, this model achieved the highest recall, F1-score and accuracy (Table 2). To validate that, we also performed McNemar's test. As a result, we confirmed our previous deduction as *BERT + PAR* statistically outperforms all the other models, even after Bonferroni correction ($p - values \ll 0.05/33$). As such, in Figure 6, we show the performance of this model for each context, as a *heat-map plot*. The rows represent the actual values, while the columns depict the predicted contexts.

Figure 6 illustrates that, in general, our best model is capable of recognising problems from each context with a high recall. Indeed, there are predictions in some classes without miss-classification such as *Computational*, *Sports*, etc. However, we can see some cases where the model made mistakes. For example, the model gets confused between the classes *Production* and *Commercial*. This may have happened because some problem statements could have come from a production context, but with focus on sales, which would be further related to the *Commercial* context. Moreover, there are some problems that are actually from the context *Production*, classified by our best model as *Date and time*. This was an unexpected result for us. After visual inspection, we noticed that some of these problems linked the efficiency of a company to the time-scale (e.g., how long a process took determined its efficiency). This is a possible explanation for such confusions within our model.

Coupled with that, according to Table 1, it is possible to notice that the class *Computational* has only a few statements. Despite this low number of problems in this context, our model is able to recognize this minority class with no errors (100% of precision and recall). Still, the class *Tax* presents the lowest number of problems in our database.

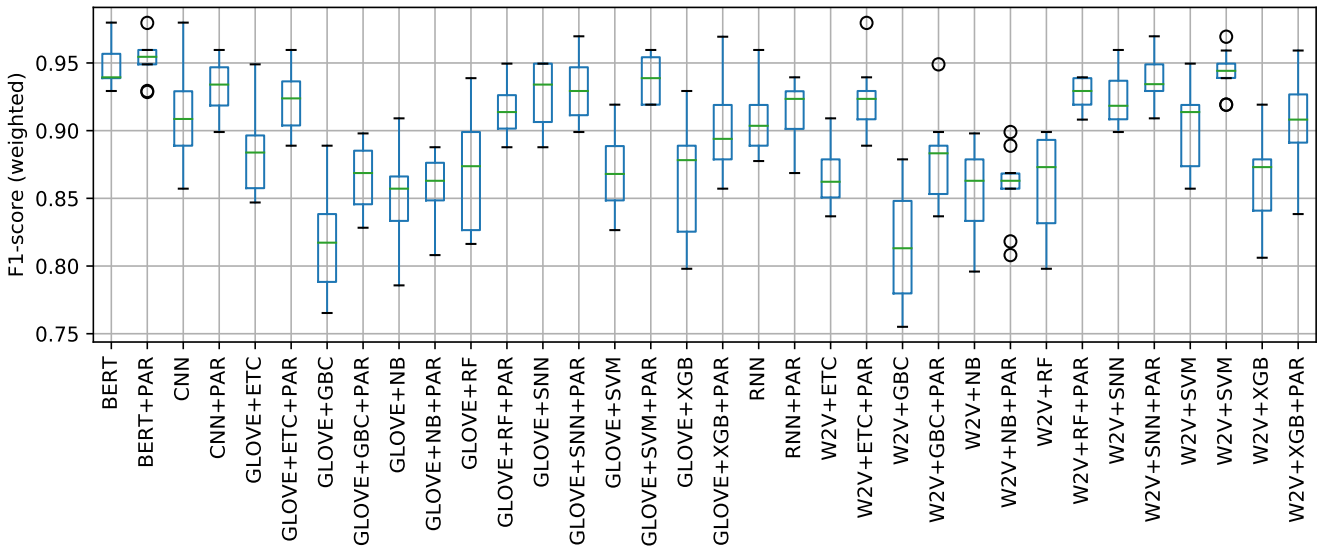

Figure 5: All results (F1-score)

But even so, our model achieved a recall of 80% in this context. Consider that the instances missclassified from the *Tax* context were allocated to *Commercial*, which makes sense, as, in some cases, these two contexts are related.

Although the model achieved a high recall (95%) in the context of *Games*, instances that the model was not able to recognise were spread through multiple contexts (*Commercial*, *Date and time*, *Physics*, *Tax*, and *Mathematical*). The 2% error between *Games* and *Tax* can be explained by statements of games that comprise tariffs, e.g., when buying a certain product within the game. For example, there are statements in our data set that discuss buying products for a character, such as a battle suit. Further, the error of 1% with the class *Commercial* could be due to a reason similar to that of the class *Tax*. To illustrate, within a game, some statements comprise the purchase of products. Regarding the class *Date and time*, an explanation would be statements that address some mission that the character needs to accomplish in a specific time. Regarding the error in the classes *Physics* and *Mathematical*, it may be due to statements in games that contain speed calculation.

Another important analysis to be done occurs in the class *Research*. The model achieved a recall of 94%, whereas 6% of errors occurred in the class *Person*. One possible reason is that surveys are conducted based on a group of people. Also, there are statements in our database that contain research carried out on some characteristics of people, such as age group, education, etc.

Another interesting outcome relates to the following classes: *Banking* and *Commercial*. Note that both presented confusion errors between each other, that is, the class *Commercial* presented wrong predictions in the class *Commercial* and vice-versa. This is justified because both classes deal with statements that involve money.

Furthermore, a similar situation occurs for the classes *Health* and *Population*. Here, errors could be due to statements addressing, e.g., the growth of a virus or bacteria. Thus, results may highlight relations between these contexts.

Another interesting analysis relates to the majority class of our data set, that is, the class *Mathematical*. Note that it was possible to obtain here a 99% *recall*. Even more importantly, note that few classes have errors in this class, that is, although we are dealing with the majority class, our model can differentiate, with high precision, all classes, against this one. To illustrate, only the following classes had a confusion error with respect to this class: *Games*, *Geography* and *Commercial*. Regarding the error presented in the prediction of the class *Games*, it is an error that could be justified by questions that deal with any type of calculation, given that any form of calculation can be directly related to the mathematical context. For the *Geography* class, the error could be justified, as we have noticed the existence of statements that deal with map scale conversion. Regarding the class *Commercial*, the error could be justified by calculating the price of a certain product.

Nevertheless, we had unexpected outcomes as well. For example, it was arguably to be expected that the *Physics* class presented errors in the *Mathematical* class, given that statements that address a physical contextualisation deal with mathematical calculations. However, this does not happen. Thus, our model clearly differentiates here between even small details present in the statement of each context.

In other words, although there is an error in the classification of some instances in the classes, most of these errors can be easily justified. This may suggest that the statements worked on in this research have multi-contextualisation, that is, a statement can address more than one context. However, what happens in practice is that one context is predominant, and the prediction of our model reflects this. Still, it is

Table 2: Classification performance of the predictive models (Pr: precision; Re: recall; F1: F1-score; Acc: accuracy).

Model	Pr(Macro)	Pr(weighted)	Re(Macro)	Re(weighted)	F1(Macro)	F1(weighted)	Acc
GLOVE+RFC	95%	88%	78%	87%	84%	87%	86.8%
GLOVE+RFC + PAR	94%	92%	86%	92%	89%	91%	91.6%
GLOVE+ETC	95%	90%	80%	88%	86%	88%	88.3%
GLOVE+ETC+PAR	95%	93%	87%	92%	90%	92%	92.3%
GLOVE+XGBC	91%	87%	74%	87%	79%	86%	86.5%
GLOVE+XGBC+PAR	92%	91%	82%	90%	85%	90%	90.2%
GLOVE+GNB	90%	86%	78%	85%	82%	85%	85.0%
GLOVE+GNB + PAR	88%	87%	82%	86%	84%	86%	86.7%
GLOVE+SVM	80%	87%	71%	87%	75%	86%	86.8%
GLOVE+SVM+PAR	95%	94%	89%	94%	91%	94%	93.7%
GLOVE+GBC	79%	83%	68%	82%	72%	81%	81.7%
GLOVE+GBC+PAR	83%	87%	77%	87%	79%	86%	86.6%
GLOVE+KC	91%	93%	89%	93%	90%	93%	92.8%
GLOVE+KC+PAR	91%	93%	90%	93%	90%	93%	93.1%
W2V+RFC	95%	88%	78%	86%	84%	86%	86.1%
W2V+RFC+PAR	94%	93%	87%	93%	90%	93%	92.7%
W2V+ETC	95%	88%	79%	87%	85%	87%	86.8%
W2V+ETC+PAR	95%	93%	87%	92%	90%	92%	92.3%
W2V+XGBC	92%	87%	76%	86%	82%	86%	86.4%
W2V+XGBC+PAR	91%	91%	86%	91%	87%	91%	90.7%
W2V+GNB	90%	87%	78%	86%	82%	86%	85.7%
W2V+GNB+PAR	88%	87%	81%	86%	84%	86%	85.9%
W2V+SVM	85%	90%	79%	90%	82%	90%	90.2%
W2V+SVM+PAR	**96%**	**95%**	91%	94%	93%	94%	94.3%
W2V+GBC	77%	82%	69%	81%	73%	81%	81.3%
W2V+GBC+PAR	83%	88%	78%	88%	80%	88%	87.8%
W2V+KC	91%	92%	89%	92%	90%	92%	92.4%
W2V+KC+PAR	93%	94%	91%	94%	92%	94%	93.9%
KT+CNN	94%	91%	84%	91%	88%	91%	90.8%
KT+CNN+PAR	92%	93%	90%	93%	91%	93%	93.2%
KT+(RNN+CNN)	86%	91%	86%	91%	85%	91%	90.8%
KT+(RNN+CNN)+PAR	89%	91%	87%	91%	88%	91%	91.4%
BT+BERT	93%	**95%**	91%	**95%**	92%	**95%**	94.7%
BT+BERT+PAR	94%	**95%**	**92%**	**95%**	**93%**	**95%**	**95.2%**

potentially useful to further analyse this problem as a multi-contextual prediction task.

7. LIMITATIONS

One of the major limitations of this paper is related to data set size. Although we have a significant number of problems, in the case of some contexts there is a small number of instances, due to the quantity of classes in our multi-classification problem. To address this limitation, we used cutting-edge NLP techniques to produce new instances on the training set, using contextual paraphrases.

Moreover, our original problem descriptions were in Portuguese and hence, when we translated them to English, this may have introduced some errors from our automatic data processing. However, this was counter-balanced by the availability of the most cutting-edge NLP processing tools for the various steps involved in our pipeline, which were not available for the Portuguese language.

In addition, this research worked with introductory topics to computer programming. It is thus less clear if the methodology applies to more advanced topics of programming. For example, database disciplines may need a different approach. However, the holistic pipeline we propose can guarantee that the right method can outperform the others, thus ensuring area appropriateness.

Another limitation arises from undergraduate courses that do not have programming in their curriculum. Although it is clear that in this research several courses may use programming for some activities, not all of them have programming topics in the curriculum. To illustrate, although our data set presents health issues that can be applied to the medical or nursing courses, unfortunately these undergraduate courses do not have programming topics in their curriculum. This may however change in the future, with the rise of the ubiquitousness of computing, and thus this research may have wider relevance and impact than originally envisioned.

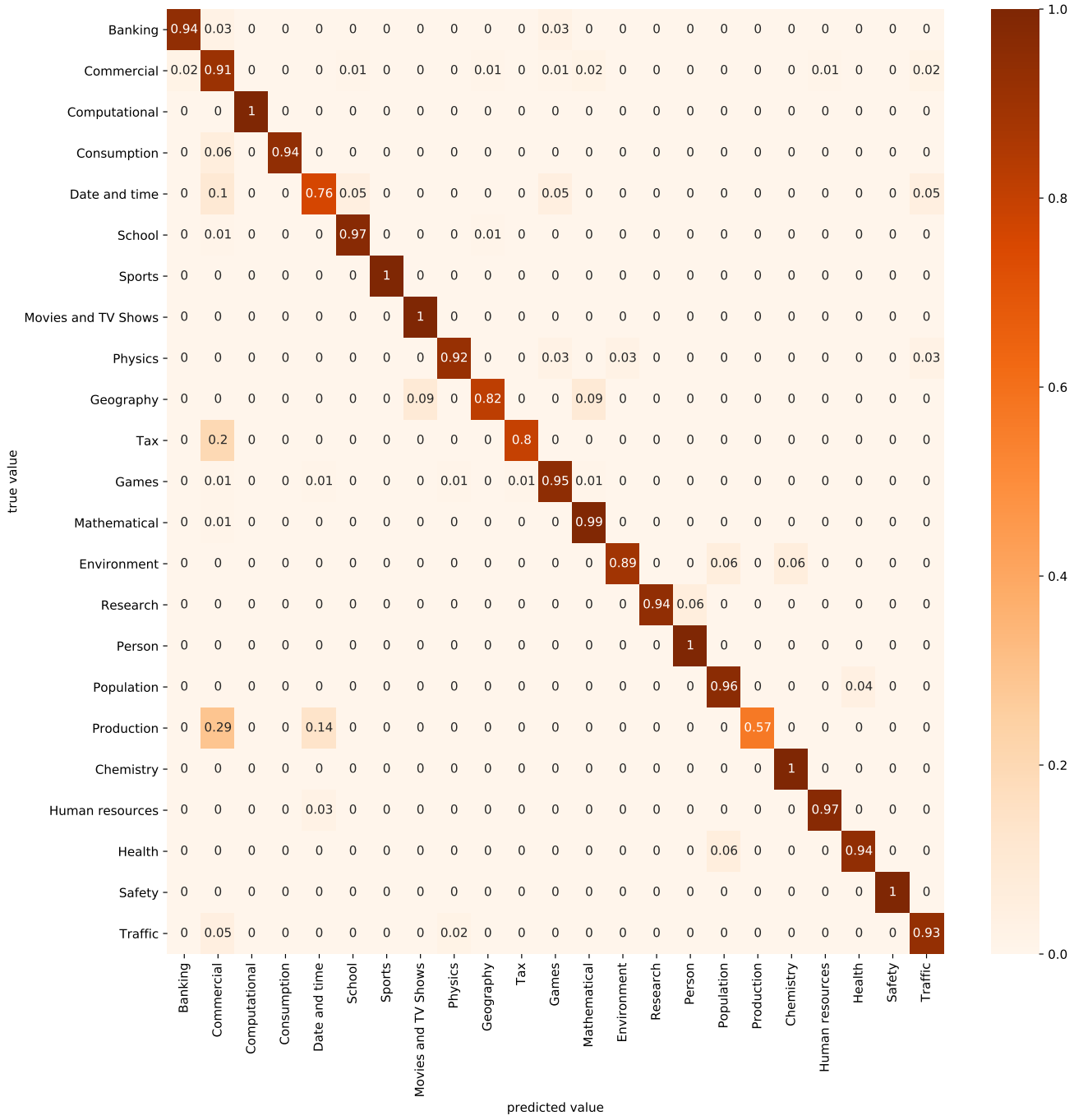

Figure 6: BERT with Paraphrasing

8. CONCLUSION AND FUTURE WORKS

According to the results obtained and illustrated in this research, we can conclude that paraphrasing of the minority classes boosts results, that is, it was able to make predictive models more accurate and with greater recognition capacity, regardless of which NLP was used, that is, Glove, Word2vec, BERT, etc.

In addition, our work was able to achieve a performance with high precision and a high recognition rate for all 23 classes proposed in this article. That is, our best model, which is based on the BERT technique with paraphrase-balancing, was able to achieve an accuracy of 95.2% with a minimal error rate, which is no more than 4.8%.

With that, the first step to *generate personalised problem lists, according to the context of the undergraduate course*, was taken. We have additionally provided a *new context taxonomy for problems*, as well as a *comprehensive evaluation pipeline methodology for context-based personalisation*

of problem lists.

As future work we intend to further evaluate the effect of the personalised programming problem assignments using our method to detect the subject matter. Thus, we can explore if the performance of the non-CS students will be affected when solving problems related to their courses.

In addition, three new experiments can be performed to analyse the generalisation power of our method. The first is to repeat the procedure on other online judge problem collections, but still at an introductory programming discipline level. The purpose of this experiment is to verify how generalisable our approach is across educational settings different from ours. We believe, nevertheless, that choices such as the programming language used in teaching CS1 will not be a factor that will prevent similar outcomes.

As a second experiment, we would repeat the procedure with more advanced programming topics, to analyse if the method can be applied to these more complex types of topics. For example, disciplines such as data structures may be a research target. Finally, we envision to adapt our pipeline to perform automatic classification of the programming problems in terms of the topics used in the CS1 courses (*Sequential*, *Composite conditional structures*, *Chained conditional structures*, *Repeating structures by condition*, *Repeating structures by counting*, *Vectors and Strings* and *Matrices*). Such a pipeline would be useful for several applications, such as for problem recommendation, automatic annotation, amongst others.

Concluding, we believe that the automatisation of the classification of statements by contexts is extremely relevant for several reasons, among which we highlight: i) statements which students are already familiar with can help in the process of engagement and learning; ii) students will find it easier to understand the relevance of programming in their professional lives; iii) teachers can use this automatisation to generate personalised lists, which would facilitate their work, since it would be too much work to select these problems manually, in addition to which it could lead to human error and iv) students could use this automatisation to select problems to which they are used to, facilitating their process of learning a certain programming topic.

Acknowledgements

This research, carried out within the scope of the Project for Education and Research (SUPER), according to Article 48 of Decree nº 6.008/2006 (SUFRAMA), was partially funded by Samsung Electronics of Amazonia Ltda, under the terms of Federal Law nº 8.387/1991, through agreements 001/2020 and 003/2018, signed with Federal University of Amazonas and FAEPI, Brazil.

9. REFERENCES

[1] T. Aljohani, F. D. Pereira, A. I. Cristea, and H. T. Oliveira. Prediction of users' professional profile in moocs only by utilising learners' written texts. In *International Conference on Intelligent Tutoring Systems*. Springer, 2020.

[2] R. Artstein and M. Poesio. *Inter-coder agreement for computational linguistics*. Educational and Psychological Measurement 20(1):37-46, 2008.

[3] V. Athavale, A. Naik, R. Vanjape, and M. Shrivastava. Predicting algorithm classes for programming word problems. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 84–93, 2019.

[4] J. Cohen. A coefficient of agreement for nominal scales. Educational and Psychological Measurement 20(1):37-46, 1960.

[5] R. E. De Castilho, J.-C. Klie, N. Kumar, B. Boullosa, and I. Gurevych. Linking text and knowledge using the inception annotation platform. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 327–328. IEEE, 2018.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

[8] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[9] F. Dwan, E. Oliveira, and D. Fernandes. Predição de zona de aprendizagem de alunos de introdução à programação em ambientes de correção automática de código. *Simpósio Brasileiro de Informática na Educação-SBIE*, 28(1):1507, 2017.

[10] L. Echeverría, R. Cobos, L. Machuca, and I. Claros. Using collaborative learning scenarios to teach programming to non-cs majors. *Computer Applications in Engineering Education*, 25(5):719–731, 2017.

[11] S. Fonseca, E. Oliveira, F. Pereira, D. Fernandes, and L. S. G. de Carvalho. Adaptação de um método preditivo para inferir o desempenho de alunos de programação. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 30, page 1651, 2019.

[12] L. Galvão, D. Fernandes, and B. Gadelha. Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 27, page 140, 2016.

[13] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[14] I. Govender. The learning context: Influence on learning to program. *Computers & Education*, 53(4):1218–1230, 2009.

[15] V. T. Norman and J. C. Adams. Improving non-cs major performance in cs1. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, page 558–562, New York, NY, USA, 2015. Association for Computing Machinery.

[16] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[17] F. Pereira, E. Oliveira, D. Fernandes, L. S. G. de Carvalho, and H. Junior. Otimização e automação da predição precoce do desempenho de alunos que utilizam juízes online: uma abordagem com algoritmo genético. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 30, page 1451, 2019.

[18] 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 *International Conference on Artificial Intelligence in Education*, pages 67–72. Springer, 2019.

[19] F. D. Pereira, E. H. Oliveira, D. Fernandes, and A. Cristea. Early performance prediction for cs1 course students using a combination of machine learning and an evolutionary algorithm. In *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*, volume 2161, pages 183–184. IEEE, 2019.

[20] 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. *British journal of educational technology.*, 2020.

[21] Y. Qian, S. Hambrusch, A. Yadav, and S. Gretter. Who needs what: Recommendations for designing effective online professional development for computer science teachers. *Journal of Research on Technology in Education*, 50(2):164–181, 2018.

[22] J. M. C. RAABE, A. L. A.; SILVA. Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos. pages 2326–2335, 2005.

[23] B. L. Santana and R. A. Bittencourt. Increasing motivation of cs1 non-majors through an approach contextualized by games and media. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Oct 2018.

[24] A. Stefik and S. Siebert. An empirical investigation into programming language syntax. *ACM Transactions on Computing Education (TOCE)*, 13(4):1–40, 2013.

[25] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal. A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, 51(1):1–34, 2018.

[26] A. S. Zanini and A. L. A. Raabe. Análise dos enunciados utilizados nos problemas de programação introdutória em cursos de ciência da computação no brasil. In *Anais do XXXII Congresso da Sociedade Brasileira de Computação, XX WEI – Workshop sobre Educação em Computação, Curitiba*, 2012.

[27] 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 Transactions on Information Systems (TOIS)*, 36(3):1–33, 2018.