Estimating the Difficulty of Programming Problems in Brazilian Portuguese with Few Examples: An Evaluation of BERTimbau Embeddings

João Pedro M. Sena[1], **Ellen Francine Barbosa**[1], **André G. Santos**[2], **Julio C. S. Reis**[2]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
13.566-590 – São Carlos – SP – Brasil

[2]Departamento de Informática – Universidade Federal de Viçosa (UFV)
36.570-970 – Viçosa – MG – Brasil

joao.sena@usp.br, francine@icmc.usp.br, andresantos@ufv.br, jreis@ufv.br

Abstract. *Programming practice is a fundamental step in learning to program. In this context, estimating the difficulty of programming questions is crucial for effective adaptive learning. Some approaches address this problem by analyzing students' past performance, but this can be impractical as sometimes only the problem text is available. For this reason, this study evaluates the use of semantic embeddings of exercise text (QDET) extracted with BERTimbau, a state-of-the-art BERT model pre-trained for Brazilian Portuguese, in a context where only a small sample is available for training, a task referred to as few-shot learning. The results show that the embeddings generated by BERTimbau do not effectively capture the difficulty of problems based solely on textual information. This emphasizes the need for models trained specifically for this task.*

1. Introduction

Programming problems play a crucial role in student learning, especially in programming courses where hands-on exercises are essential for deep understanding [Rodrigues et al. 2022, Silva et al. 2023]. In this context, estimating the level of difficulty of questions and exercises – whether in programming or in another subject – is crucial for effective adaptive learning. This approach allows for greater personalization of the learning experience as it helps to select an exercise that matches the learner's abilities, which has proven to be very beneficial in education [Chen et al. 2005]. However, manual assessment of difficulty is both time and resource consuming. Two common approaches are to analyze students' psychometric data to determine the level of difficulty or to consult experts who know the context well. In both cases, the process requires a lot of time and significant human input, which is not feasible in certain situations [Benedetto et al. 2023].

Other approaches have also been explored in the literature, such as analyzing students' attempts or submitted code [Silva et al. 2022]. However, these methods can be impractical when problems are newly created as such data is not readily available, delaying their effective use in educational settings [de Freitas Júnior et al. 2020]. In contrast, using the text of the instruction provides an estimate of the difficulty of a problem as soon as it is formulated, allowing earlier application in an educational context. Several studies have applied different techniques for Question Difficulty Estimation from the text (QDET) or "question calibration" to address this challenge [Benedetto et al. 2023].

DOI: 10.5753/sbie.2025.12253

In this scenario, the need for semantic representations in text classification (whether for QDET or other applications) has driven the development of various embedding methods. These embeddings are multi-dimensional vector representations that capture important text features and enable effective classification in different domains [Benedetto et al. 2023]. Since the concept of difficulty can vary depending on the educational context, the use of embeddings can save the need to retrain a new model, especially in contexts with languages such as Brazilian Portuguese. A prominent approach is BERT (Bidirectional Encoder Representations from Transformers) [Devlin et al. 2019], a Large Language Model (LLM) based on the Transformer architecture, which has been widely adopted for text representation [da Costa et al. 2023]. These models are particularly useful in scenarios with extremely limited data (which are common in real-world applications). They use pre-training to improve classification performance even when the dataset is small, a task known as few-shot learning [Parnami and Lee 2022].

In this work, a dataset of 109 programming problems with statements in Portuguese was used for difficulty estimation based on their textual content (code is available in Section 6). BERTimbau, a modern pre-trained BERT model for Brazilian Portuguese [Souza et al. 2020], focusing on the evaluation of its embeddings. A dataset was constructed from the manual labeling of problems by an expert, according to different programming topics, and different models performed the classification based on the embedding representations extracted by BERTimbau. In sum, the results suggest that the pre-trained model does not effectively capture the difficulty of these problems in its representations. This emphasizes the need for models that have been trained or tuned specifically for this task, as well as the importance of database availability.

The rest of the paper is organized as follows: Section 2 discusses related work. Sections 3 and 4 describe the dataset and methodology, respectively. Subsequently, Section 5 presents the results, and Section 6 concludes the study by outlining limitations and possible future directions.

2. Related Work

There are some previous studies that focus on QDET in conjunction with programming exercises using pre-trained models. For example, [Benedetto et al. 2021] suggests the use of transformer models such as BERT and DistilBERT to estimate the difficulty of multiple-choice questions from the text. They pointed out the subjectivity and computational cost of these classical methods and suggested fine-tuning the pre-trained natural language processing models to capture textual nuances that affect the difficulty of the question. The results show that BERT achieves the lowest error compared to baseline methods. The study also highlights the efficiency of DistilBERT when computational resources are limited, as it achieves similar performance to BERT with fewer parameters.

In [Zhou and Tao 2020], the authors propose MTBERT (Multi-task BERT), a BERT-based model for predicting the difficulty of programming questions on online platforms, overcoming the limitations of traditional methods based on manual evaluation or complex feature extraction. Their work addresses the lack of reliable information about the difficulty of the questions and the inefficiency of traditional models, such as BiLSTM, which require extensive pre-processing and long training times. The solution leverages BERT's multi-task fine-tuning and shares embedding representations between different

datasets to improve generalization and reduce noise. The results show that MTBERT achieves higher accuracy and is trained faster, highlighting its ability to capture the textual information relevant to the difficulty. Also, the model uses accuracy rate as an indicator of difficulty and shows a correlation between complex text and lower accuracy.

Furthermore, the study presented in [Moreira et al. 2024] proposes an unsupervised approach to group programming exercises into thematic clusters based on the source code of the solutions to support teachers in selecting personalized teaching material. The central problem is the lack of automatic organization of exercises by topic in educational platforms, which makes it difficult to create lists aligned to the level of students. The method uses techniques including BERT to generate contextual embeddings of the source codes, followed by dimensionality reduction with t-SNE and clustering via K-means. The results show that BERT outperforms conventional approaches in capturing semantic patterns and enables the formation of coherent, clusters validated by experts. The analysis revealed that the clusters reflect topics such as data structures, functions and basic operations. This shows the potential of the technique to automate the categorization of exercises and optimize pedagogical planning. The study suggests future extensions to estimate the difficulty level of questions based on the identified clusters.

This work complements efforts to investigate the potential of machine learning techniques in automatically predicting the difficulty of questions, including programming exercises. However, **the dataset used here contains programming exercises in Brazilian Portuguese**, a language in which the estimation of the difficulty of programming exercises has not been investigated in previous work. Although [Moreira et al. 2024] works with a dataset in Brazilian Portuguese, the approach focuses on categorizing the exercises by topic without estimating their difficulty. This represents a direct contribution to the evaluation of a technique that is widely used in other languages. In addition, [Benedetto et al. 2021] and [Zhou and Tao 2020] focus on evaluating the tested models, training or fine-tuning them. Here, **the evaluation refers specifically to the task of learning with few shots**, where no large set of cataloged data is available. Considering that categorizing this data can be costly, **this is likely to be a scenario closer to the context of use for ordinary users** who do not have access to large datasets. Finally, **the code is made available** in Section 6, allowing for a **new reproducible benchmark** for the community interested in this problem.

3. Dataset

In this section, we present details of the dataset explored in this study, including the labeling process and an exploratory analysis.

3.1. Context of data

The dataset was built from a set of programming problems used in an introductory computer programming course at *Universidade Federal de Viçosa (UFV)*[1] (Federal University of Viçosa), a public university in Brazil[2], offered in the first semester of its Computer Science course. The aim of this subject is to introduce students to the initial concepts of computer programming, which include the development of problem-solving skills and

[1] `https://www.ufv.br/`

[2] As this is a Brazilian subject, the exercises are in Brazilian Portuguese.

programming logic (especially in structured programming). This subject, which lasts approximately 15 weeks, consists of three weekly classes, each lasting approximately two hours. In general, two of the classes consist of explanatory activities, while the other consists of practical activities that focus on the *C++* language to teach the main concepts. In this subject, the practical activities are not graded directly, but it is important to emphasize that there are about three practical exams, which account for approximately 30% of the students' final grade, and a similar dynamic comes into play.

The practical activities classes are conducted in a laboratory with one computer for each student, with the help of a teaching assistant, by making available approximately five programming problems in an automatic exercise grading system (known as online judge) called BOCA[de Campos and Ferreira 2004], similar to programming contests such as the International Collegiate Programming Contest (ICPC)[3]. Also, the problems follow a format similar to those used in programming contests, presented through a PDF file available on the system. The problem descriptions detail a programming task that involves one or more topics of the subject and should allow the student to develop a code that solves it. The descriptions, in general, consist of an introduction explaining the problem context and task, two sections specifying the format of the inputs and expected outputs, and a section with some examples of inputs and outputs. An example of a translated problem can be seen in Figure 1. This study is part of a larger project on programming exercise recommendation. The use of data in this study was approved by the Ethics Committee of the *Universidade Federal de Viçosa* under project number CAAE 75327023.0.0000.5153.

3.2. Labeling

The current professor has been teaching this subject for more than ten years and has gained deep insight into the challenges faced by students. Based on this experience, he has identified nine key topics that summarize the course content: repetition structures ("Repetition"), conditional structures ("Conditional"), functions ("Function"), arrays or homogeneous data structures ("Array"), mathematical applications ("Math"), matrix manipulation ("Matrix"), sorting algorithms ("Sorting"), strings or sequences of characters ("String") and structs or heterogeneous data structures ("Struct").

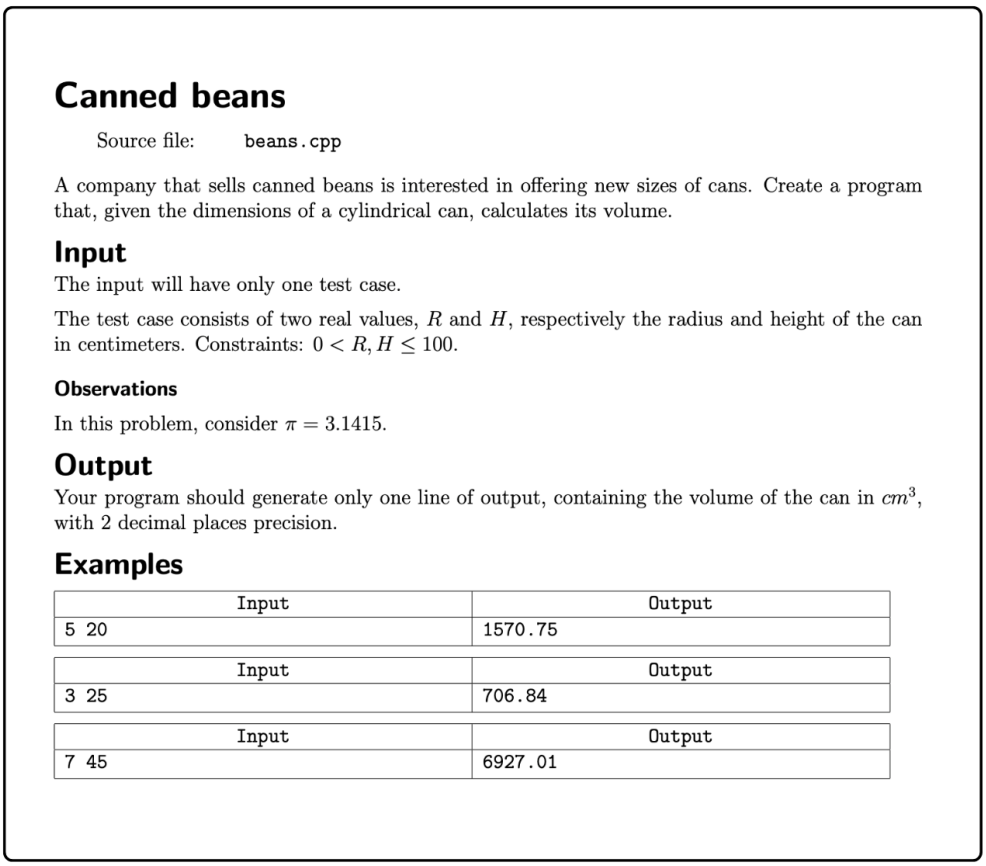
Over the years, various problems have been used in the course, each addressing one or more of these topics at different difficulty levels. To classify them, the professor assigned each problem a difficulty rating for each topic, using four possible levels: 0 ("out"), 1 ("easy"), 2 ("medium"), and 3 ("hard"). A difficulty level of 0 means that the problem does not cover that topic, while levels 1, 2, and 3 indicate increasing difficulty.

For example, the problem illustrated in Figure 1 requires students to write a program that calculates the volume of a cylinder given the radius of its base and height. Since this problem primarily involves mathematical calculations, it was classified as level "easy" in the "Math" topic and level "out" in all others. As classes progress, problems become more complex, gradually incorporating additional topics.

Over the years, a set of 109 programming problems has been obtained, including those used in practical classes and exams. To ensure standardization, the professor generates PDF files using LaTeX[4], which allows for the complete extraction of text content.

[3]More about ICPC: `https://icpc.global/regionals/abouticpc`.

[4]LaTeX is a software system for typesetting documents.



Canned beans

Source file: `beans.cpp`

A company that sells canned beans is interested in offering new sizes of cans. Create a program that, given the dimensions of a cylindrical can, calculates its volume.

Input

The input will have only one test case.

The test case consists of two real values, R and H, respectively the radius and height of the can in centimeters. Constraints: $0 < R, H \leq 100$.

Observations

In this problem, consider $\pi = 3.1415$.

Output

Your program should generate only one line of output, containing the volume of the can in cm^3, with 2 decimal places precision.

Examples

Input	Output
5 20	1570.75

Input	Output
3 25	706.84

Input	Output
7 45	6927.01

Figure 1. PDF image of an example problem (translated into English).

In addition, the professor has labeled each problem according to the nine predefined topics and has assigned a unique numerical identifier (ID) to each. Part of the labeling table is shown in Table 1, where the first column is the problem ID and the subsequent columns indicate the difficulty level for each topic.

Table 1. Illustration of the labeling table.

ID	Repetition	Conditional	Function	⋯	String	Struct
0	0 (out)	0 (out)	0 (out)	⋯	1 (easy)	0 (out)
1	0 (out)	0 (out)	0 (out)	⋯	1 (easy)	0 (out)
⋮	⋮	⋮	⋮	⋱	⋮	⋮
107	1 (easy)	3 (hard)	1 (easy)	⋯	1 (easy)	0 (out)
108	1 (easy)	2 (medium)	1 (easy)	⋯	0 (out)	0 (out)

3.3. Exploratory analysis

First, we conduct an exploratory analysis to gain a deeper understanding of the dataset and its content. The analysis included (i) the distribution of problems classified with each label, (ii) the correlation between length and difficulty, and (iii) a word cloud representing the content of all programming problems.

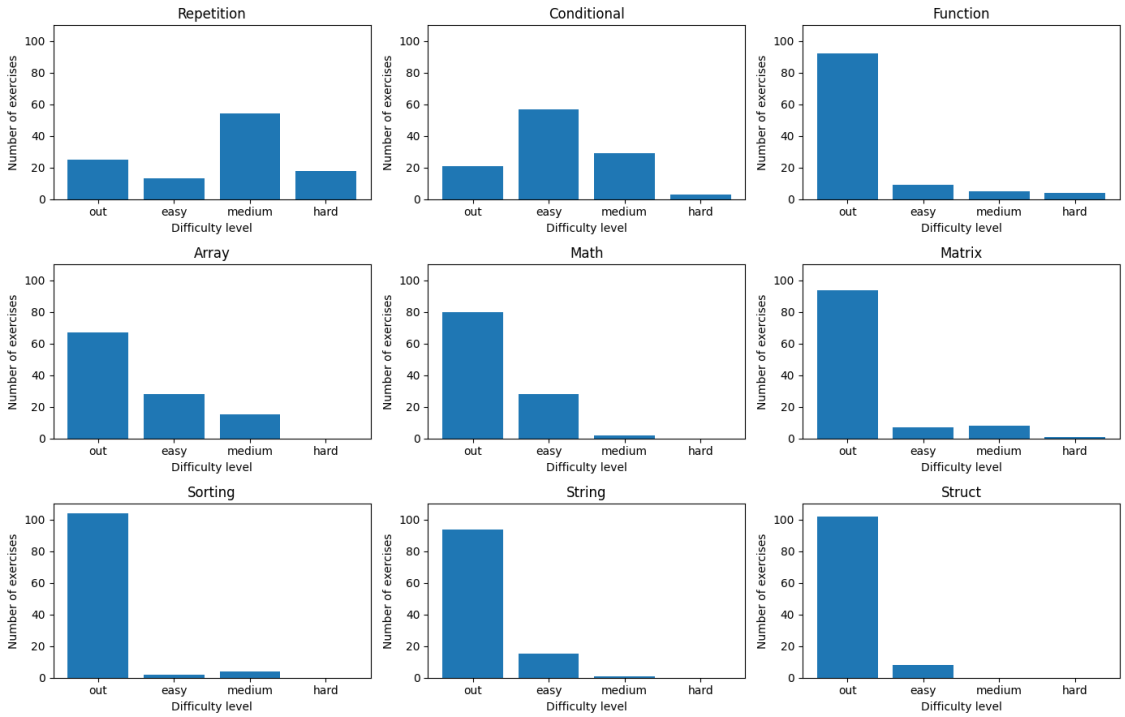


Figure 2. Distribution of difficulty labels in each topic.

For this analysis, the text was first split into tokens, which, in this case, are words. The process of token extraction included a pre-processing step to remove non-textual content (numbers, punctuation and mathematical expressions), conversion of all characters to lowercase, and removal of words that are used frequently but carry little semantic meaning (stop words). During this pre-processing, only the tokens that have real semantic value are retained. It is worth noting that this pre-processing step was also used throughout the experiment.

Figure 2 shows the distribution of problems on each topic, together with the respective difficulties. This reflects a characteristic of the educational context in which different topics may have different priorities and require varying levels of practice, resulting in a different number of problems for each topic. It is noticeable that, for many topics, a large number of problems are not included (labeled "out"), such as "Sorting", "String" and "Struct". This unbalance is expected, as these topics are taught later in the course. In addition, some topics do not have problems labeled "medium" or "hard", probably because they are suggested for an introductory programming level.

This distribution shows an unbalanced dataset, i.e., the number of examples in each category is not evenly distributed. This imbalance can negatively impact classification performance, as the model may have very few (or no) examples representing certain levels of difficulty for specific topics. With fewer examples, the model may not be able to generalize effectively to classify these topics.

Further important analysis can be found in Figure 3, which shows a set of points representing the problems. The x-axis represents the length of the problem (number of tokens), while the y-axis represents the total difficulty (sum of the difficulties for each

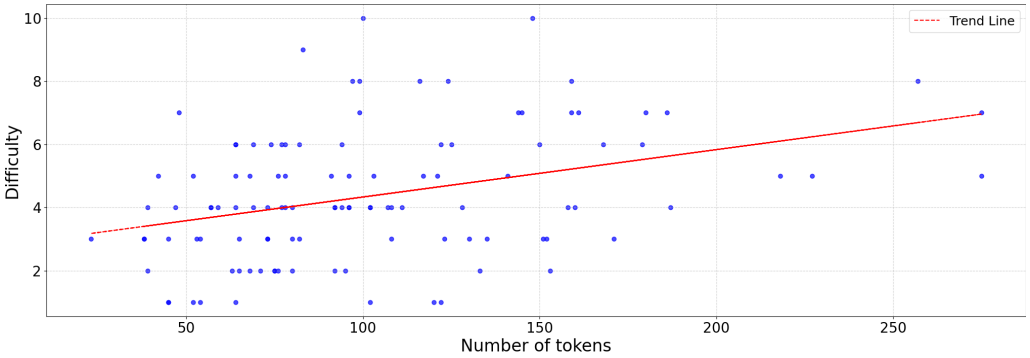


Figure 3. Correlation between length and difficulty.

topic). A trend line created by linear regression between these two variables is also included. Although the trend line indicates a positive correlation, there are many outliers, suggesting that difficulty levels cannot be accurately represented by problem length alone.

Finally, a word cloud was also used. A word cloud visually represents the meaning of tokens by displaying more frequent terms in a larger size. By analyzing all tokens across multiple programming problems, it provides valuable insight into their overall content. Figure 4 shows the word cloud generated from the dataset, where common tokens such as "*programa*" ("program"), "*linha*" ("line"), "*deve*" ("should"), and "*número*" ("number") prominently stand out, as they appear frequently in most problems. These words are typically used to describe the problem, as seen in sentences like "*Seu **programa deve** ler cada **linha** com um **número** n_i*" (in Portuguese) ("Your **program should** read each **line** with a **number** n_i"), but they are not highly discriminative. Furthermore, problem descriptions usually introduce a brief scenario that justifies the need for a program without specifying the techniques required to solve the problem. This suggests that analyzing word frequency alone is not sufficient to assess the complexity of a problem and that the vector representation must capture not only term frequency but also, to some extent, the techniques used to solve the problem that affect its difficulty.

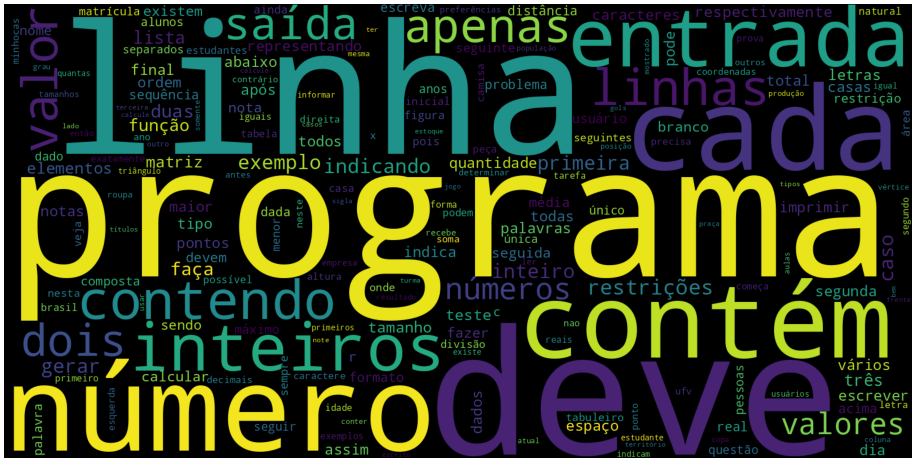


Figure 4. Word cloud of the tokens obtained in the problem statements.

4. Methodology

The classification of the problem difficulty was performed through a three-step process: (i) extracting the exercises' text from LaTeX files, (ii) generating semantic vectors (embeddings) from the problem text, and (iii) classifying the embeddings using various machine learning models. The implementation was done in Python, using libraries such as HuggingFace's Transformers [Wolf et al. 2020][5] and Scikit-learn [Pedregosa et al. 2011][6].

4.1. Classification

First, the text was extracted from the corresponding LaTeX files. During this step, a pre-processing phase was applied to remove noisy content such as LaTeX configuration settings and commands. This process assumes that such elements have no impact on the difficulty of the problems and should, therefore, be discarded. Additionally, certain LaTeX commands, including mathematical expressions, images, and equations, were replaced with special tags (e.g., "[MATHMODE]", "[IMAGE]", "[EQUATION]") to ensure that the model considers their presence when generating vector representations.

For semantic vector extraction, two distinct approaches were employed: BERTimbau embeddings and TF-IDF embeddings. For BERTimbau, the CLS strategy was chosen. This approach uses the special "[CLS]" token (stands for "Classification"), which captures the individual token representations generated by BERTimbau into a single vector representing the overall meaning of the entire text. However, BERT imposes a 512-token limit per sequence, so the tokens that exceeded this limit were ignored, resulting in a final programming problem representation of 768 dimensions. Also, as a baseline embedding strategy, Term Frequency-Inverse Document Frequency (TF-IDF) was employed. TF-IDF is a numerical statistic reflecting the importance of a word within a document relative to a collection of documents (corpus) [Salton 1991]. For this study, the TF-IDF vectors were generated from the preprocessed problem texts. This provides an interpretable baseline for comparison with the more complex BERTimbau embeddings. Since the BERTimbau model uses words to capture context, there is no additional pre-processing step, unlike TF-IDF. In the case of BERTimbau, the full text has been preserved by simply removing the LaTeX configuration and settings, so that the text corresponds to what can be seen in the corresponding PDF file. In the case of TF-IDF, stopwords were removed, tokens were lowercased and only the words were retained.

Given that each problem can have up to four difficulty labels per topic, this classification task is framed as a multitask classification problem. In this setup, there are N topics, each with a specific number of labels $(M_1, M_2, ..., M_N)$. In this case, there are nine topics ($N = 9$), and each topic can have up to four difficulty levels ($M_i \leq 4$). To align the model's output with the expected format, a separate instance of the classification model was trained for each topic.

The classification models were chosen based on prominent approaches found in the literature for text classification using embeddings [da Costa et al. 2023]: Random Forest (RF) [Breiman 2001], Support Vector Machine (SVM) [Joachims 1998] and Multi-

[5] https://huggingface.co/docs/transformers/index
[6] https://scikit-learn.org/

layer Perceptron (MLP) [Taud and Mas 2017]. Each of these models was evaluated for its ability to classify across the different topics.

4.2. Models evaluation

Since this is a multitask classification problem, the evaluation was performed both independently for each topic and simultaneously across all topics. To achieve this, first, one problem was removed from the dataset to keep at least one example for label. After this, the dataset was split into two subsets with a 75%-25% proportion, with a train set of 81 examples and a test set of 27 examples, totaling 108 problems used in the experiments. All models were trained and tested using these same sets. Specially, in the training step, a grid search strategy was used for parameters tuning after normalizing the data evaluated by a cross-validation using, each time, one sample for training and the rest for evaluation (leave-one-out method [Cawley and Talbot 2003]). Then, the testing was done with the best models.

The dataset split was carefully designed to account for topics with a very low number of examples in certain difficulty levels. By ensuring a sufficiently large number of test examples, the evaluation could more accurately reflect performance across minority labels. To guarantee this, a search strategy was used to keep the proportion of labels close in both sets during partitioning, resulting in at least one example of each label present in both the training and test sets.

For evaluation, the F1 score was chosen as the primary metric [Baeza-Yates et al. 1999]. The F1 score is the harmonic mean of precision and recall, balancing both the proportion of true positive predictions and the proportion of false negatives. Given that each topic can have up to four difficulty levels, the F1 score was computed separately for each label and then averaged using a simple mean. This approach ensures that all labels contribute equally to the final evaluation, preventing majority classes from dominating the results.

5. Experimental Results

The evaluation assessed the performance of Support Vector Machine (SVM), Random Forest (RF), and Multi-layer Perceptron (MLP) models for classifying problem difficulty across various topics, utilizing both BERTimbau and TF-IDF embeddings. Table 2 presents the F1 score results for each model and embedding combination, with the highest score in each row highlighted in bold.

The results in Table 2 indicate that MLP achieved the best overall performance across both embeddings. In most topics, this was particularly evident with BERTimbau embeddings, though in some cases TF-IDF also produced strong results. Using MLP, the highest scores were obtained for nearly all topics, with the exception of “Function”, where the best model surpassed it by only 0.01. Specifically, BERTimbau embeddings combined with MLP achieved the best F1 scores in seven out of nine topics, reaching a mean F1 score of 0.62, the highest among all model–embedding configurations.

Notably, the topics “Math”, “Matrix”, and “String” stand out as outliers with exceptionally high scores of 0.83, 0.93, and 1.00 (respectively), all achieved with the MLP model. Interestingly, despite being extremely imbalanced, these topics yielded the best

Table 2. F1 score for each model in each topic.

Topic	RF		SVM		MLP	
	BERT	**TF-IDF**	**BERT**	**TF-IDF**	**BERT**	**TF-IDF**
Repetition	0.25	0.35	0.25	0.24	**0.39**	**0.39**
Conditional	0.30	0.25	0.23	0.32	0.42	**0.45**
Function	**0.23**	**0.23**	**0.23**	**0.23**	0.22	0.22
Array	0.46	0.46	0.46	0.40	**0.48**	0.46
Math	0.36	0.39	0.36	0.39	**0.83**	0.38
Matrix	0.31	0.31	0.31	0.31	0.82	**0.93**
Sorting	0.32	0.32	0.32	**0.66**	**0.66**	**0.66**
String	0.89	0.73	0.73	0.73	**1.00**	0.81
Struct	0.47	0.47	0.47	**0.73**	**0.73**	**0.73**
Mean $\pm$ s.d.	0.40 ± 0.19	0.39 ± 0.14	0.37 ± 0.15	0.44 ± 0.19	$\mathbf{0.62 \pm 0.24}$	0.56 ± 0.22

performances, indicating that imbalance alone does not fully explain performance differences. For “String”, high scores were observed across different embeddings and models, suggesting that the statements in this topic contain easily distinguishable linguistic characteristics. The best result was achieved with BERTimbau, which captures semantic context, in combination with MLP, which models complex relationships between features and labels. “Math” also performed best with MLP and BERTimbau, reinforcing the idea that both contextual information and more sophisticated feature–label interactions are important. This is intuitive, since classifying mathematical problems may require not only recognizing the type of calculation involved but also linking it to difficulty, which often follows more complex patterns. In contrast, “Matrix” achieved its best score with TF-IDF, though BERT also delivered competitive results. This suggests that, for this topic, surface-level word frequencies are already highly discriminative, reducing the need for deeper semantic representation.

On the other hand, the topic “Function” exhibited the weakest performance. It was the only case where MLP was not the best-performing model, though the variation across models was minimal, with all scores around 0.22 and differing by no more than 0.01. Problems in this topic can be considered a special case: they are often small variations of previous problems, with the additional requirement of implementing the solution within a function. As a result, the textual descriptions show only minor changes, while the critical difference lies in the code specification. This suggests that code itself may be a more reliable source of information for difficulty classification in this context, rather than the problem statement alone.

For most topics, except “Math”, the scores obtained with BERT and TF-IDF embeddings using MLP were competitive. In some cases, such as “Repetition”, “Function”, “Sorting”, and “Struct”, the results were even identical. This suggests that BERTimbau did not always provide deeper or more useful semantic representations for capturing problem difficulty, since a simple keyword-based method like TF-IDF often performed similarly. Even in topics with more balanced distributions, such as “Repetition”, TF-IDF was nearly as effective. This indicates that, for certain topics, key terms alone may carry sufficient information to approximate difficulty, reducing the benefits of contextual embeddings. Nevertheless, overall performance remained low for most topics, reinforcing the need to explore alternative approaches for practical applications.

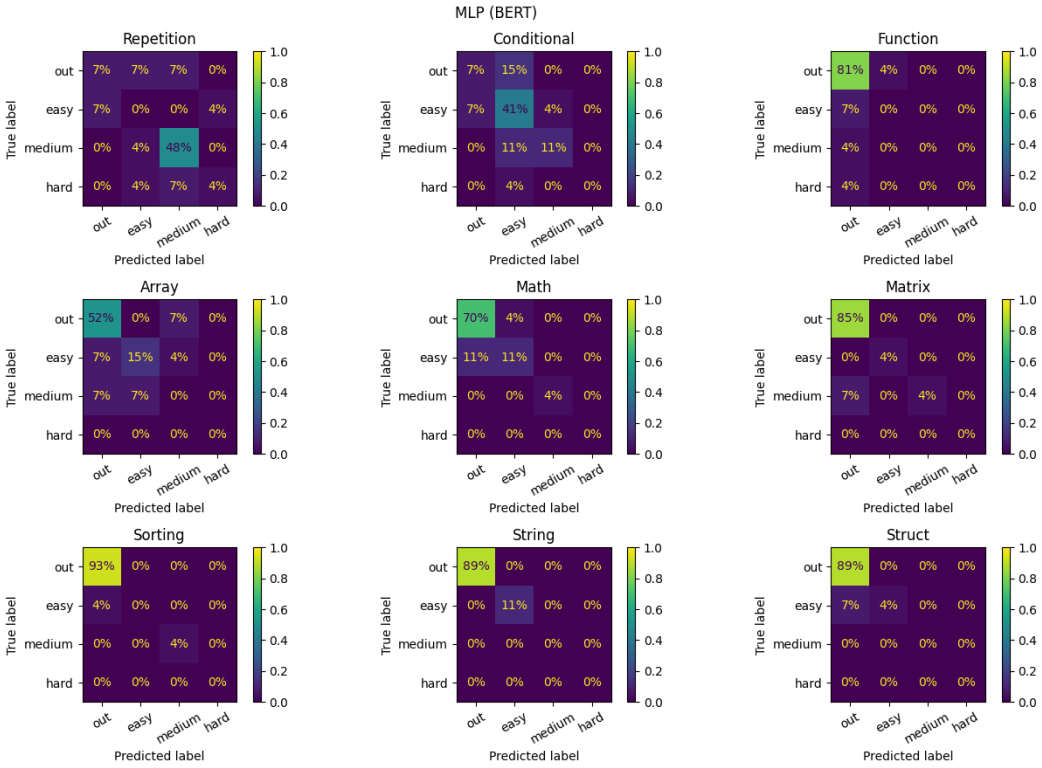


Figure 5. Confusion matrix for MLP with BERTimbau embeddings.

Figure 5 presents the confusion matrix for MLP with BERTimbau embeddings, the best-performing model across the main portion of topics. The confusion matrix highlights important aspects of MLP's behavior. A key observation is its tendency to prioritize the majority label (generally "out"), even in topics where the distribution of labels is relatively balanced, such as "Repetition" and "Conditional". This suggests that label imbalance does play a role, but the embeddings' limited ability to represent problem features is also a contributing factor.

Last, for "Repetition" and "Conditional", the confusion matrix shows a reasonable percentage of correct predictions for non-majority classes. For example, in "Conditional", 7% of problems were correctly classified as "out" and 11% as "medium", while "easy" remained the majority label. However, despite this ability to generalize, the F1 scores for these topics were not as high as for others with more irregular distributions, such as "Math" and "String". This indicates that lower F1 performance cannot be explained by imbalance alone, but may also result from embeddings that are suboptimal for capturing difficulty in those topics. Some categories may have difficulty more strongly tied to linguistic cues already captured by BERTimbau. For example, in "Math", terms related to geometric calculations may signal greater complexity than those describing basic arithmetic in the easier exercises.

6. Conclusion and Future Work

This study investigated the classification of 109 programming problems using embeddings from BERTimbau, a pre-trained language model for Brazilian Portuguese. These

embeddings served as input for multi-task classification models, with performance evaluated using the F1 score across all topics.

The results indicate that BERTimbau struggled to capture problem difficulty, even in topics with relatively balanced label distributions, though it achieved notably good performance in certain cases. The confusion matrix further reveals that, even for the best-performing model, predictions are biased toward the most frequent labels. This suggests that while data imbalance amplifies the issue, the embeddings alone may not sufficiently encode problem difficulty when relying exclusively on textual content.

The dataset size represents a limitation for the analysis, even when accounting for the distribution of exercises across difficulty levels within each topic. Nevertheless, the objective here is not to test model training, but rather to assess the effectiveness of pre-trained embeddings, which can be meaningfully evaluated even with few examples. In real-world educational contexts, difficulty labels are often closely tied to subject matter, and a semantic representation that fails across different labeling configurations is unlikely to be useful. Moreover, it is unrealistic to assume that teachers and students applying difficulty assessment techniques will have access to large datasets. For this reason, it is essential to evaluate such techniques under data-limited scenarios.

A valuable intermediate step could be to assess the ability of Large Language Models (LLMs) to solve programming problems or whether they can abstract the knowledge required to solve them. If this is the case, embeddings that can be used in generative models to solve the problem could also better capture the difficulties that students face in overcoming these challenges. This work is expected as future work.

Also, given these results, training a specialized model for this task is a promising direction. However, the current dataset may not be large enough for effective training, which emphasizes the importance of expanding the dataset. The plan is to first build a larger dataset to train a specialized model for the classification of programming problems in Brazilian Portuguese. Once this model is trained, it can be fine-tuned for similar tasks so that it can perform well in scenarios with limited examples, as mentioned earlier.

Resources

The code and instructions for requesting data are available at `https://github.com/jpmedras/qdet_programming_ptbr`.

Acknowledgements

The authors would like to express their gratitude, since this study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. The authors also acknowledge financial support from FAPEMIG (Fundação de Amparo à Pesquisa do Estado de Minas Gerais) and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico). Julio would also like to thank the INCT TILD-IAR (process number 408490/2024-1) for their support. Finally, the authors are especially grateful to researchers Carlos Eduardo Paulino Silva and Lucas N. Ferreira for their valuable comments and insightful discussions at different stages of this project.

References

Baeza-Yates, R. R., Ribeiro-Neto, B., et al. (1999). *Modern information retrieval*. ACM Press.

Benedetto, L., Aradelli, G., Cremonesi, P., Cappelli, A., Giussani, A., and Turrin, R. (2021). On the application of transformers for estimating the difficulty of multiple-choice questions from text. In Burstein, J., Horbach, A., Kochmar, E., Laarmann-Quante, R., Leacock, C., Madnani, N., Pilán, I., Yannakoudakis, H., and Zesch, T., editors, *Proc. of the Workshop on Innovative Use of NLP for Building Educational Applications*, pages 147–157.

Benedetto, L., Cremonesi, P., Caines, A., Buttery, P., Cappelli, A., Giussani, A., and Turrin, R. (2023). A Survey on Recent Approaches to Question Difficulty Estimation from Text. *ACM Comput. Surv.*, 55(9).

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Cawley, G. C. and Talbot, N. L. (2003). Efficient leave-one-out cross-validation of kernel fisher discriminant classifiers. *Pattern recognition*, 36(11):2585–2592.

Chen, C.-M., Lee, H.-M., and Chen, Y.-H. (2005). Personalized e-learning system using item response theory. *Computers & Education*, 44(3):237–255.

da Costa, L. S., Oliveira, I. L., and Fileto, R. (2023). Text classification using embeddings: a survey. *Knowledge and Information Systems*, 65(7):2761–2803.

de Campos, C. P. and Ferreira, C. E. (2004). Boca: um sistema de apoio a competições de programação. In *Workshop de Educação em Computação (WEI)*.

de Freitas Júnior, H. B., Pereira, F. D., de Oliveira, E. H. T., de Oliveira, D. B. F., and de Carvalho, L. S. G. (2020). Recomendação automática de problemas em juízes online usando processamento de linguagem natural e análise dirigida aos dados. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 1152–1161.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proc. of the European Conference on Machine Learning (ECML)*, pages 137–142.

Moreira, J., Silva, C., Santos, A., Ferreira, L., and Reis, J. (2024). Abordagem não-supervisionada para inferência do tópico de um exercício de programação a partir do código solução. In *Anais do Workshop sobre Educação em Computação (WEI)*, pages 842–853.

Parnami, A. and Lee, M. (2022). Learning from few examples: A summary of approaches to few-shot learning. *arXiv*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Rodrigues, G., Monteiro, A. F., and Osório, A. (2022). Introductory programming in higher education: A systematic literature review. *OASIcs, Volume 102, ICPEC 2022*, 102:4:1–4:17.

Salton, G. (1991). Developments in automatic text retrieval. *Science*, 253(5023):974–980.

Silva, C. E. P., Solano, J. L. S., dos Santos, A. G., and Reis, J. C. S. (2023). Previsão de reprovações em disciplinas introdutórias de programação: Um estudo em um ambiente de correção automática de códigos. In *Anais do Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 1524–1535.

Silva, E. S., Carvalho, L. S., de Oliveira, D. B., Oliveira, E. H., Lauschner, T., de Lima, M. A., and Pereira, F. D. (2022). Previsão de indicadores de dificuldade de questões de programação a partir de métricas do código de solução. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 859–870.

Souza, F., Nogueira, R., and Lotufo, R. (2020). BERTimbau: pretrained BERT models for Brazilian Portuguese. In *Brazilian Conference on Intelligent Systems (BRACIS)*.

Taud, H. and Mas, J.-F. (2017). Multilayer perceptron (mlp). *Geomatic approaches for modeling land change scenarios*, pages 451–455.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Huggingface's transformers: State-of-the-art natural language processing. In *arXiv*.

Zhou, Y. and Tao, C. (2020). Multi-task bert for problem difficulty prediction. In *Proc. of the International Conference on Communications, Information System and Computer Engineering (CISCE)*, pages 213–216.