2023 IEEE 8th International Conference for Convergence in Technology (I2CT)
Pune, India. Apr 7-9, 2023

Autograding of Programming Skills

Nakka Narmada
Department of Computer Science and Engineering
Amrita School of Computing
Bangalore, India
bl.en.p2dsc21018@bl.students.amrita.edu

Peeta Basa Pati
Department of Computer Science and Engineering
Amrita School of Computing
Bangalore, India
ORC ID: 0000-0003-2376-4591

2023 IEEE 8th International Conference for Convergence in Technology (I2CT) | 979-8-3503-3401-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/I2CT57861.2023.10126211

***Abstract*— To evaluate a learner's knowledge of programming language skills, assessments are given. Grading of these is usually done manually which not only is tedious but prone to error due to repetition and fatigue. In this work, we employ pre-trained language models to perform automated grading of "C" programming language. Embeddings from different transformers on pre-assessed codes are used as feature vectors to train a wide range of regressors for the scoring task. Root-mean-square error (RMSE) is the metric utilized to compare the scores of these regressors. It's observed that embeddings from T5-model with CatBoost regressor gives the least error around 15%.**

Keywords—Programming languages, autograding, Transformer embeddings, T5, CatBoost regressor.

I. Introduction

Assignments are an important part of any learning process. They are used to evaluate a learner's progress and understanding of the material. They provide an opportunity for an individual to practice and apply the concepts and skills they have learnt. It can also help instructors identify areas where learners may be struggling and provide additional support as needed. They help the learners develop skills, including problem-solving, critical thinking, and the ability to communicate their ideas effectively. Many assignments are designed to mimic real-world problems or scenarios.

Most of the programming courses use assignments as they help learners practice and develop their skills and provide a way for instructors to evaluate their progress and understanding [11]. It also allows learners to gain experience solving problems that are similar to those they may encounter in a professional setting. They are an important part of the learning process for learners studying computer science and related fields.

There are several ways in which programming assignments are assessed & evaluated [1-3,11].

- Automated grading: Programming assignments can be graded automatically using tools such as test runners or code checkers. These tools can check the output of a program against a set of expected results and provide a grade based on how closely the output matches the expected results.
- Peer review: In some cases, learners may be asked to review and evaluate the work of their peers. This can be done through code review, where learners review each other's code and provide feedback on its quality and efficiency.
- Manual grading: Some programming assignments may require more subjective evaluation and may be graded manually by an instructor, teaching assistant or a Subject Matter Expert(SME). This may involve reviewing the code and output of a program, as well as evaluating the design and overall approach taken by the learner.
- Combination of methods: It is also possible to use a combination of these methods to evaluate programming assignments. For example, an assignment may be graded automatically using a test runner, and then the instructor may manually review the code and output to ensure that it meets the requirements of the assignment.

The method of evaluation used for programming assignments will depend on the specific goals and the preferences of the instructor.

Currently, the Automated grading system uses specialized software tools that check the output of a program against a set of expected results and provide a grade based on how closely the output matches the expected results. The accuracy of automated grading depends on the quality and completeness of the test cases used to evaluate the code. If the test cases are incomplete or flawed, the grading may not be accurate. Also, they may not be able to fully evaluate more subjective aspects of a submission, such as the design or overall approach taken by the learner.

There are several benefits associated with usage of autograding system for programming assignments.

- Speed: It can be much faster than manual grading, especially for large classes or assignments with many submissions. This can be especially helpful when there is a high volume of submissions or when instructors have limited time to grade assignments.
- Consistency: It can help ensure that all submissions are graded consistently and fairly, as the grading is based on objective criteria rather than subjective judgment.
- Cost: It can save time and reduce the workload for instructors, particularly for large classes or assignments with many submissions. However, there may be initial costs associated with setting up and implementing an automated grading system, and some studies have found that the use of automated grading may require additional resources, such as technical support or training.

Autograding can be an effective tool for evaluating programming assignments, particularly when the grading criteria are clear and well-defined. However, it is important to carefully consider the limitations and potential trade-offs associated with using autograding.

In this work, we experiment with usage of deep learning based language models to vectorize the code content and employ various regression models to mimic the scoring pattern obtained from expert instructors. This Autograding system is employed for C programming language. The novelties of present work are described below.

- Expansion of the C-programming language database [10] with more code blocks created by introducing error.
- Evaluation of pre-trained T-5 model for bi-class classification to separate codes which are accepted from the ones that are rejected.
- Employ T-5, BERT & CodeBERT models to vectorize the code blocks. These vectors are used through regression models to assign evaluation scores.

Through the experiments conducted in this work, we have tried to answer the following research questions.

- Can pretrained T5 model, trained with natural language content, be employed to perform bi-class classification of accepted code from rejected code?
- Can embeddings derived from pre-trained models like BERT, T5, CodeBERT be used with statistical regressors to mimic human experts' evaluation pattern?

II. Literature Survey

To understand the process of grading it is necessary to understand how a human perceives or judges a solution. The approach varies from human to human. To maintain some consistency rubrics were suggested. Srikant and Aggarwal [1-3] suggested that assessing a computer programme according to a rubric is a key feature of an automated grading system. Such a rubric often links a score (quantitative measure) to the programmer's capacity for problem-solving. Srikant and Aggarwal [3] present the first machine learning method for autonomously grading programmes using a novel language of features that captures the distinctive components that human experts look for when deciding on a grade. These attributes, which are connected to a suggested problem-independent rubric, capture the program's functioning. Comparatively speaking, this outperforms a simple keyword-counts model. To train a model for each question, they calculate how far a code is from the good codes. On a 5-point Likert scale, each fresh response to the same topic is graded using this method. They further enhanced this strategy to create a question independent model which automatically creates structurally invariant features, normalises them and then uses these features in joint learning across questions [4]. This is then used to predict labels for answers to hypothetical questions. Another way of approaching autograding is to find the similarity between two codes, it is a widely studied area. A method of code abstraction to identify source code similarities was put forth by Park et al. [5]. However, this method results in the loss of crucial knowledge regarding the conditions and logic applied in a programme. It is more beneficial in situations where there is a lot of code, and the abstraction offers the necessary standards. A strategy that aims to harmonise all the programmes' formats was introduced by Wang et al. [6]. Comparing a programme to a set of accurate model programmes is necessary for the automatic grading of that programme. This method results in an accurate grade. It is not, however, scalable. standardization gets more challenging, and errors are more likely to appear as the amount of the code grows. Using these techniques, a question independent model based on bag of words was developed [7]. For each piece of code using the Clang libraries, the control flow graph (CFG) and the data flow graph (DFG), which tracks the movement of variables, were combined. The bag of words method is used to turn the enhanced graph of each response into a conventional format. These phrases all refer to a code feature. The matching sub-bag of codes from the good set is compared to the sub-bag of each code.

Another approach is assessing structural similarity according to a reference solution [8]. They outline an approach based on machine learning (ML) that maps the system's predicted results to the scores given by the rubric. The method is empirically tested using programming assignments submitted by students taking a Python programming course at the college level. Since the authors utilise machine learning techniques and consider various characteristics of the programme structure, the computational complexity of the approach scales with the size of the programme. However, the system's intended use case is the assessment of modestly sized programming issues in introductory programming classes, therefore the complexity has little bearing. Discrete Wavelet Transform (DWT) was suggested to evaluate source code files at various resolutions and to find plagiarism in C, C++, and Java source code files [9]. Static code analysis was used to detect malicious blocks in the software [10]. Adding compiler transformations, the efficiency of static analysis of code was further enhanced [12]. Eye-tracking could also be used to analyze the code [13]. An unsupervised word sense disambiguation (WSD) algorithm is suggested to improve grading process of essays.[14] Md.Rabin and Md. Alipour [15] have worked on neural code analyers to test if transformations in a code impacted the way neural program analyzers performed. They observed that large size of codes and multiple transformations did impact the performance of the neural code analyzers.

Most work related to source code analysis focus on using features extracted based on the lexical structure of the code. With the help of ASTs, CFGs and DFGs these features are extracted [1-4], [7-8]. This creates computational complexity as the size of the code could compromise the system capability. When dealing with similar tasks in NLP like Automated Grading of answer scripts or essays, many state of the model transformer systems like BERT, GPT, T5 are used. These systems create embeddings of the input text which is then used to train models to perform the desired task. To utilize similar techniques, we explored intos source code embeddings. Currently the area of source code embeddings is still in nascent stage. Code2Vec is a machine learning model that was developed by researchers at Cornell University and is specifically designed for natural language processing (NLP) tasks on source code [16]. Code2Vec is based on the Word2Vec model, which extends the Word2Vec model to work with source code by representing code tokens (such as variables, function names, and control structures) as words in a "code vocabulary." The Code2Vec model consists of a neural network with several layers, including an input layer, an output layer, and hidden layers. The input layer maps the code tokens to the code vocabulary, and the output layer maps the code tokens to their corresponding embeddings. They utilize AST to create the input sequence. Building on BERT, a widely used NLP model that uses a transformer architecture and a guided attention mechanism to analyze text, CodeBERT was developed which is specifically designed for natural language processing (NLP) tasks on source code [17]. CodeBERT is trained on a large dataset of source code and natural language comments, which allows it to understand the relationships between the code and the comments and to perform various NLP tasks on the code. It was designed to perform tasks like Code Classification, Code Completion, Code Translation and Code Summarization.

III. Dataset

The dataset used for this study is borrowed from [7]. To this original dataset we have added 263 more samples. Each of the programming tasks could be answered in multiple ways. A common grading scale (Table 1) was used as a rubric to rate these answer codes manually by the SME. The distribution of marks for the dataset is presented in Fig. 1. In the scoring strategy, the 10 mark denotes the best solution while the 1 mark represents the worst solution to the question.

TABLE I. GRADING SCALE RUBRIC USED BY SME'S FOR SCORING CODE BLOCKS.

Description	**Grade**
Incorrect logic and incomprehensible code	1-2
Incorrect logic and comprehensible code	3-4
Hints of correct logic with correct data structures	5-6
Correct logic and correct structuring but bad quality code	7-8
Correct logic and correct structuring with good quality code	9-10

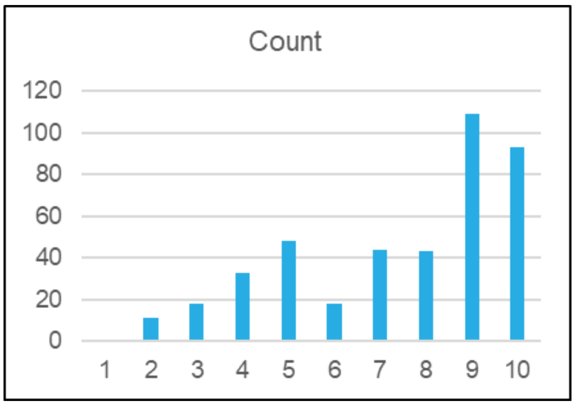

Fig. 1. Distribution of grades for the dataset of codes.

IV. System Architecture

In this section, we describe the models used for this study. We have worked with a pre-trained model in the initial stage and then in the later stage, we used transformer embeddings to train regression models as presented in Fig. 2.

For this experiment the original dataset was used to perform binary classification using T-5 pre-trained model. After performing classification on the original dataset, it was expanded. On the new dataset, we use pre-trained

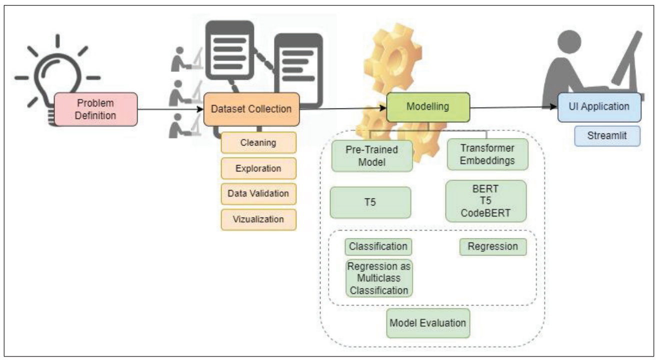

Fig. 2. System Flowchart

transformers BERT, T-5 and CodeBERT and get the respective code embeddings. These embeddings are passed as feature vectors to regressor models such as Decision Tree, KNN, SVR, XGBoost, CatBoost, Neural Network. These models are then evaluated using RMSE and MAE. The best model is then saved and a dashboard is created using Streamlit.

T5

T5 (Text-To-Text Transfer Transformer) is based on the transformer architecture which capture complex relationships between the input and output data. The model uses multiple layers of encoder and decoder blocks, which consist of a self-attention layer and a feed-forward layer. The self-attention layer allows the model to attend to different parts of the input and output sequences and capture the dependencies between the tokens. The feed-forward layer applies a non-linear transformation to the data. T5 uses a combination of character and word embeddings to represent the input and output data, which allows it to handle a wide range of input and output data, including code and structured data.

Transformer Embeddings

Transformer embeddings are vector representations of words, sentences, or other input elements that are learned using a transformer model. Transformer models are a type of deep learning model that are designed to process sequential data, such as natural language text or time series data. They are based on the transformer architecture. Transformer models learn embeddings by processing the input data through multiple layers of attention and transformation, which allows them to capture complex relationships between the input elements and generate rich and expressive embeddings. Transformer models have been widely used in NLP tasks such as machine translation, language modeling, and text classification, and they have achieved state-of-the-art performance in many of these tasks.

Transformer embeddings can be used in a variety of machine learning tasks, such as classification, clustering, and similarity matching. They can also be used as input to other machine learning models, such as neural networks or decision trees, to improve the performance of these models on sequential data tasks. These embeddings are generally more expressive and informative than semantic or statistical embeddings, which are based on hand-crafted features or pre-defined statistical properties of the input data.

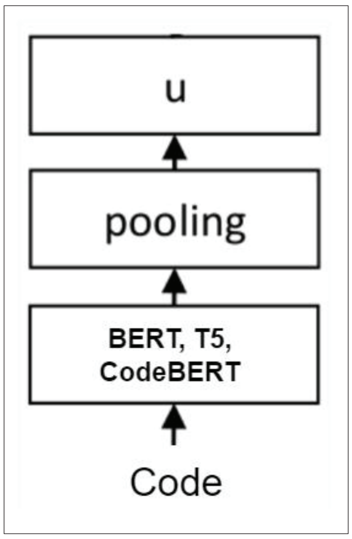

Fig. 3. Transformer Embedding Architecture

Transformer embeddings can be extracted in several ways:

- Using a pre-trained transformer model
- Fine-tuning a pre-trained transformer model
- Train a transformer model from scratch

In our work we have used transformer embeddings extracted from pre-trained BERT, T5 and CodeBERT models.

Both BERT and T5 are trained on natural languages while UniXcoder is trained on the CodeSearchNet dataset as shown in Table 2.

TABLE II. TRANSFORMER MODELS

Transformer	Model	Dataset
BERT	all-MiniLM-L6-v2	1 billion pairs
T5	t5-base	C4 Dataset
CodeBERT	unixcoder-base-nine	CodeSearchNet dataset + 1.5M NL-PL pairs of C, C++ and C#

V. RESULTS AND DISCUSSION

In this work we have attempted to address two important research questions. This section presents the results of experiments carried out to answer both these questions. Following sections present the questions, results and inferences obtained from the results.

Q1: Can a pretrained T5 model, trained with natural language content, be employed to perform bi-class classification of correct code from code with error?

To test this we have used a pre-trained T5-base model which uses a single model architecture for all of its NLP tasks, and the specific task is determined by the input and output data. We have used the subset of 200 C programming codes along with their assigned marks and labels. The T5 model was used to perform two tasks – Binary Classification and Regression(using multilabel classification).

In Binary Classification, the dataset is being classified into accepted and wrong. Accepted being the codes the that have the appropriate logic and pass the minimum requirement to pass while wrong being the codes that did the pass the minimum requirement. We found that the False negatives and False positives is relatively high as shown in Table 3. On further analysis, we find the accuracy, F1 score, precision and recall of the model on training set for binary classification to be 0.99 showing that the model was overfitting. Its performance on test data dropped to 0.48. On using stratification on the same dataset to ensure that the ratio of distribution of the two classes remain the same in train and test set, we observe that the performance of the model improves. Its accuracy, precision, F1-score and recall increased to 0.68 from 0.48 on stratification as observed in Table 4.

In general, we can observe that in most assignments most learners pass the minimum requirement. Hence, usually there is an inherent bias in real-life scenario.

TABLE III. CONFUSION MATRIX

		Test Set		Train Set	
		Accepted	Wrong	Accepted	Wrong
Before Stratification	Accepted	15	9	95	1
	Wrong	12	5	0	64
After Stratification	Accepted	21	3	96	0
	Wrong	10	7	1	63

TABLE IV. EVALUATION OF BINARY CLASSIFICATION

	Before Stratification		After Stratification	
Evaluation Metrics	Training Set	Test Set	Training Set	Test Set
Accuracy	0.99	0.48	0.99	0.68
Precision	0.99	0.47	0.99	0.68
Recall	0.99	0.48	0.99	0.68
F1 Score	0.99	0.47	0.99	0.68

The regression task in T5 is achieved using multi-label classification. For this, each grade was considered as a different class. We found that this too like the Binary classification model overfit on the training dataset as seen in Fig.4 while it performed poorly on the test dataset as seen in Fig.5. This model is then evaluated using RMSE and MAE results of which are shown in Table 5.

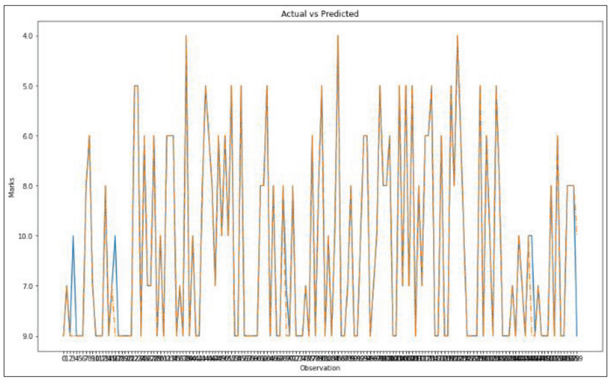

Fig. 4. Training

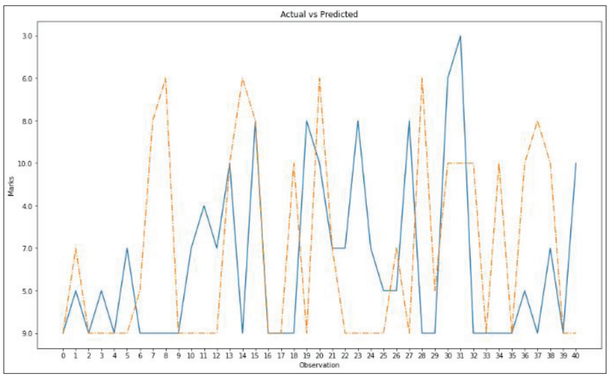

Fig. 5. Testing

TABLE V. EVALUATION OF MULTICLASS CLASSIFICATION

Evaluation Metrics	Training Set	Test Set
RMSE	1.54	2.55
MAE	0.83	1.55

Q2. Can embeddings derived from pre-trained models like BERT, T5, CodeBERT be used with statistical regressors to mimic human experts' evaluation pattern?

To achieve the task of regression, we used transformer embeddings. Each of the pre-assessed code were converted to an embedding of length 768 each. These were used as features. The total data was then split into train and test and then passed to standard machine learning and deep learning models. The results have been tabulated in Table 6. We find that there is substantial improvement in the models compared to the multiclass classification on T5. RMSE of 1.50 was achieved using CatBoost regressor with T5 transformer embeddings. We can see the CatBoost model performance in Fig. 7-9 using BERT, T5 and CodeBERT embeddings respectively. This means to say that the average difference between the predicted and actual value varies around 1.5 which indidcates a loss of 15%.

TABLE VI. EVALUATION OF REGRESSOR MODELS

	BERT		T5		CodeBERT	
	RMSE	MAE	RMSE	MAE	RMSE	MAE
Decision Tree	2.23	1.49	2.20	1.44	2.30	1.57
SVR	1.79	1.40	2.06	1.69	1.75	1.37
KNN	1.66	1.15	1.16	1.12	1.71	1.12
XGB	1.62	1.22	1.60	1.18	1.65	1.21
RF	1.62	1.26	1.54	1.20	1.56	1.21
CatBoost	1.53	1.18	1.50	1.10	1.51	1.11
NN	2.74	2.33	2.40	1.99	2.90	2.41
LSTM	2.27	1.96	2.26	1.97	2.53	2.08
GRU	2.34	2.01	2.39	2.01	2.60	2.70

When looking into the absolute differences between the actual and predicted values in the testset. In Table 7, we find that the count of observations that have the absolute difference between 0.5- 1 is maximum while using T5 embeddings while the count of observations having the maximum absolute difference greater than 2 is maximum while using CodeBERT embeddings. Based on Fig.6 this we can say that regression models using T5 and CodeBERT embeddings have performed superior to regression models using BERT embeddings.

TABLE VII. DISTRIBUTION BASED ON ABSOLUTE DIFFERENCE

Difference	BERT	T5	CodeBERT
0-0.5	29	35	32
1-2	46	41	42
>2	18	17	19

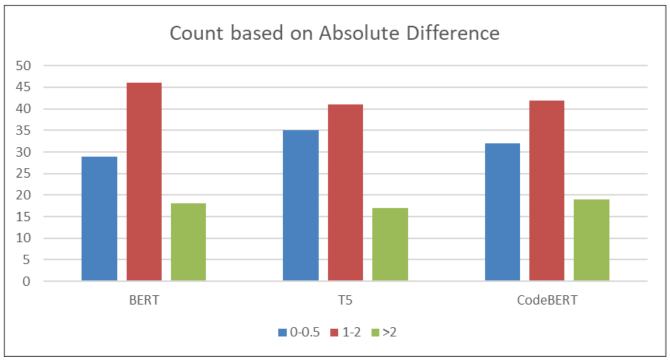

Fig. 6. Count based on absolute difference

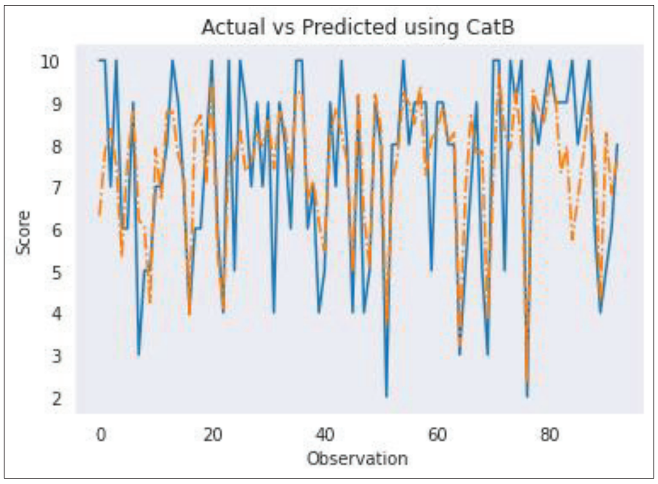

Fig. 7. BERT embedding

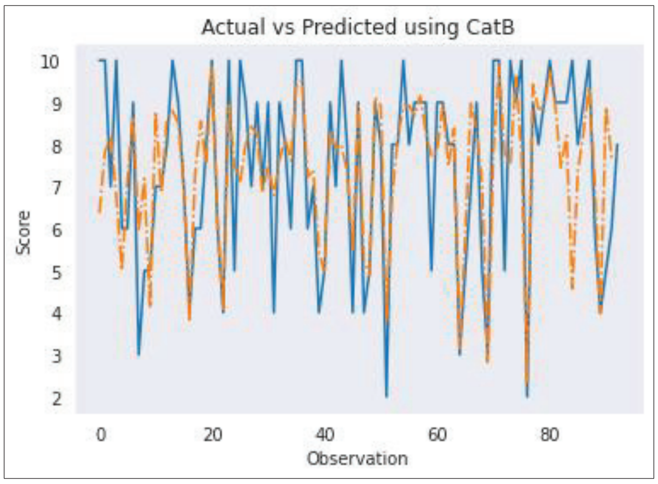

Fig. 8. T5 Embedding

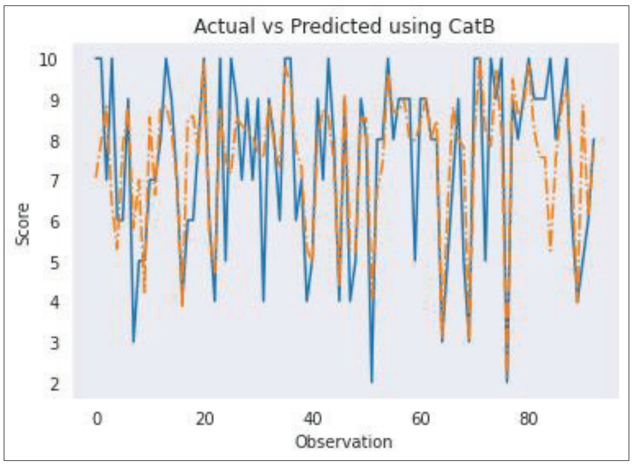

Fig. 9. CodeBERT embedding

In Fig. 7-9 we also observe that the regression model has performed equally good and bad on certain question in all three embeddings. We find that the question pertaining to finding the index of the pair with the largest difference and the index of the pair was performed well by all three embeddings. Among the three embeddings, T5 had the least difference between the predicted and actual value. The question pertaining to a game which was based on predicting the winner of the game based on conditions was underperformed by all three embeddings – in this BERT embeddings performed better over the rest. We can say that the performance of each embedding might vary based on the question as shown in Fig. 10.

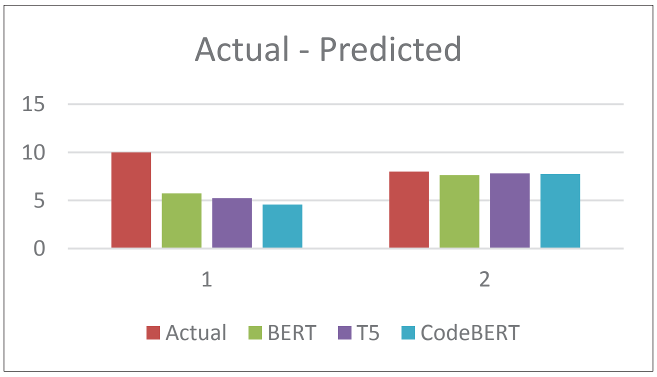

Fig. 10. Comparison of embeddings

Overall, we observe that using embeddings to train regressor models performs better when compared to directly using pretrained model like T5 for performing the scoring task of programming codes.

VI. Conclusion

Using a pre-trained NLP model like T5 may not be apt for grading programming skills. It can be limited to be used as a classification problem to check if a given code passes the minimum requirement or not. It can be used to segregate the codes into accepted and rejected classes. Embeddings can help in achieving the task of grading. With the help of embeddings, regression models can be trained which show promising results. Of the three embeddings used in the models, T5 and CodeBERT embedding can be termed superior over BERT embedding as the models trained on them could predict the scores with lesser absolute difference. Further, these embeddings can be combined to understand if a combination of the three could improve the performance of the regression models.

References

[1] V. Aggarwal, S. Srikant, V. Shashidhar. Principles for using Machine Learning in the Assessment of Open Response Items : Programming Assessment as a Case Study. 2013, NIPS Workshop on Data Driven Education

[2] S. Srikant, V. Aggarwal. Automatic Grading of Computer Programs: A Machine Learning Approach. 12th International Conference on Machine Learning Applications (ICMLA), 2013.

[3] S. Srikant and V. Aggarwal. (2014). A system to grade computer programming skills using machine learning. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1887-1896.

[4] G. Singh, S. Srikant and V. Aggarwal. (2016). Question independent grading using machine learning: the case of computer program grading. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 263-272.

[5] S. Park, S. Ko, J. Choi, H. Han, S. Cho and J. Choi. (2013). Detecting source code similarity using code abstraction. Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication, Article 74.

[6] T. Wang, X. Su, Y. Wang and P. Ma. (2007). Semantic similarity-based grading of student programs. Information and Software Technology, 49(2), 99-107.

[7] K. K. Rai, B. Gupta, P. Shokeen and P. Chakraborty, "Question Independent Automated Code Analysis and Grading using Bag of Words and Machine Learning," 2019 International Conference on Computing, Power and Communication Technologies (GUCON), 2019, pp. 93-98.

[8] Arjun Verma, Prateksha Udhayanan, Rahul Murali Shankar, Nikhila KN, and Sujit Kumar Chakrabarti. 2021. Source-Code Similarity Measurement: Syntax Tree Fingerprinting for Automated

[9] N. G. Resmi and Dr. Soman K. P., “Multiresolution analysis of source code using discrete wavelet transform”, International Journal of Applied Engineering Research, vol. 9, pp. 13341-13360, 2014.

[10] Pa Seshagiri, Vazhayil, Ab, and Padmamala Sriram, “AMA: Static Code Analysis of Web Page for the Detection of Malicious Scripts”, Proceedings of the 6th International Conference on Advances in Computing & Communications 2016, Procedia Computer Science, vol. 93. Elsevier, pp. 768-773, 2016.

[11] Sasidharan, Sarath & Joseph, Amudha. (2020). Vısual question answering models Evaluation. 1-5. 10.1109/INCET49848.2020.9154094.

[12] Thushara M. G. and Dr. Somasundaram K., “Static Analysis on Floating-Point Programs Dealing with Division Operations”, International Journal of Advanced Computer Science and Applications, vol. 10, 2019.

[13] K. R. Chandrika and Amudha J., “A fuzzy inference system to recommend skills for source code review using eye movement data”, Journal of Intelligent & Fuzzy Systems, vol. 34, no. 3, pp. 1743 - 1754, 2018

[14] Prof. Prema Nedungadi and Raj, H., “Unsupervised word sense disambiguation for automatic essay scoring”, Smart Innovation, Systems and Technologies, vol. 27, pp. 437-443, 2014

[15] Md Rafiqul Islam Rabin, Mohammad Amin Alipour, ProgramTransformer: A tool for generating semantically equivalent transformed programs, Software Impacts, Volume 14, 2022, 100429, ISSN 2665-9638, https://doi.org/10.1016/j.simpa.2022.100429.

[16] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. Code2vec: learning distributed representations of code. Proc. ACM Program. Lang. 3, POPL, Article 40 (January 2019), 29 pages. https://doi.org/10.1145/3290353

[17] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In Findings of the Association for Computational Linguistics: EMNLP 2020, pages 1536–1547, Online. Association for Computational Linguistics.