META-LEARNING FOR KNOWLEDGE TRACING IN STUDENT CODING DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Predicting whether a student will be able to solve new coding problems given past performance is an important but challenging task. Employing a traditional supervised machine learning approach to tackle this classification problem is intractable because of the large amount of data needed per student to make reasonable predictions. In this paper, we employ a meta-learning approach to solve this issue, where each task corresponds to predicting a single student's coding performance given data on that student's past performance; this allows us to exploit the benefits of shared structure that individual student's submission data would have. Our best model, a Memory Augmented Neural Network (MANN) [2] architecture which concatenates submission history embeddings to handcrafted submission features and embeddings representing coding problem concepts, achieves an accuracy of 90.5 percent and F1 score of 90.9 percent on this problem. We find that adding submission history embeddings and coding concept embeddings to our baseline model increases the accuracy by 4 percent and F1 score by 3 percent.

1 INTRODUCTION

Knowledge Tracing is the task of modelling student knowledge over time to be able to predict how students will perform in the future. This is a well established problem in computer supported education that aims to provide personalized learning experiences to each student. While various complex deep learning architectures have been used to solve knowledge tracing problems [6, 7] naively treating knowledge tracing problems as traditional supervised learning problems brings up challenges in models generalizing to unseen test examples about new students.

Meta-learning frameworks have shown promising results on such problems with sparse datasets [1], by learning how to learn classification from merely a few data examples. ProtoTransformer[1] introduced the concept of using Meta-Learning framework on student coding data. However, our task is distinct from the ProtoTransformer task in that we attempt to predict whether a student succeeds or struggles on a new coding problem, whereas the latter seeks to provide feedback on student coding submissions. While there have been several papers that try to solve knowledge tracing problems using deep learning NLP tools such as LSTMs and Transformers, there is no paper written on using a Meta-Learning framework for a knowledge tracing problem.

Other related papers focus more on how to best determine embedding representations of code. These include papers such as Code2Vec [2], which uses the attention architecture along with abstract syntax tree representation of code, to create an embedding. Another approach is CodeBert [3] which fine-tunes the classic BERT model on data from programming languages using a custom objective function. However, these papers merely focus on the embedding task for programming languages and do not employ a meta-learning framework in tackling problems.

In this paper, we use a meta-learning framework solve the knowledge tracing problem of predicting future performance on coding problems given past performance on coding problems. We therefore formulate this problem as a fewshot binary classification problem: given a handful of student coding submissions to past problems and meta-data on history of errors and performance

details, a model must quickly adapt to predict whether the student will succeed or struggle on a future problem. Our main contribution is using meta-learning for a knowledge tracing problem (which has not been done before).

The ability to predict how well a student will do on a new CS coding problem is extremely useful because it would allow educational institutions to provide informed feedback and support to students who are struggling and have a risk of failing the course. It can also enable students to learn at their own pace, being well-informed about what their future performance on problems would look like.

2 Problem Formulation

2.1 Dataset Description

We use data from the CodeWorkout dataset released through the 2nd CSEDM Data Challenge. Some of the CodeWorkout dataset is unlabeled for CSEDM challenge purposes, but in order to have the flexibility to conduct full experimental analysis in our project, we omit the unlabeled data and make use of the labeled data only. The labeled subset of data we use consists of 46k+ Java code submissions from 246 students for 50 coding problems in the Spring 2019 semester of a CS1 course at a public university, with each coding submission consisting of around 10-26 lines of code. The submission scores (% of passing unit tests) and timestamps are provided, as well as any compilation error messages. During the course, students received automated feedback from the compiler and test cases, and were allowed to submit any number of times. An example of a coding problem for the dataset is shown in Figure 1:

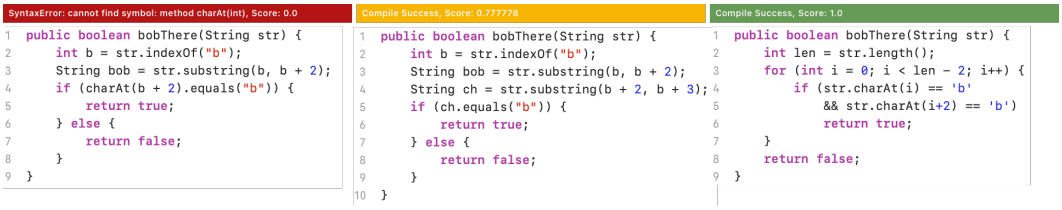

Figure 1: Sample code submissions with submission scores from 3 sampled students for a problem: *Return true if the given string contains a "bob" string, but where the middle 'o' character can be any character.*

Our labels use the same labeling strategy recommended in the 2nd CSEDM Data Challenge; that is, labels are the intersection of achieving submission correctness on a problem and requiring fewer attempts than 75% of other students who attempted the problem. This labeling strategy provides a few advantages. Firstly, it makes positive labels more rare which reduces the label skew noticeably. Further, this label definition provides semantic meaning valuable to the goal of identifying and supporting struggling students. Since students were given feedback from the coding system after each attempt, the number of submission attempts measures how much of this feedback they need to succeed, a sign that the student could use additional teaching help.

2.1.1 Meta-Learning Problem Setup

We employ a meta-learning framework to solve the binary classification problem described above, where each task corresponds to predicting a single student's coding performance given data on that student's past performance.We randomly sample 5 questions as our support set and 1 question for our query set, from a single student's submission data. We use a batch size of 64 students to sample questions and their outcomes, masking the question outcomes from the query example. Here, the dataset split is as follows: training set 148 (60%) students, validation set 61 students (25%), test set 37 students (15%). Due to label skew issues encountered while initially training our model, we altered the problem to specifically include a minimum of 2 negative labels for each sampled student's support set code submissions in order to further balance training. We additionally removed an additional 29 students who had fewer than 2 incorrect code submissions in total. This sampling

change improves convergence in training. The final dataset split is: training set 126 (60%) students, validation set 52 students (25%), test set 31 students (15%).

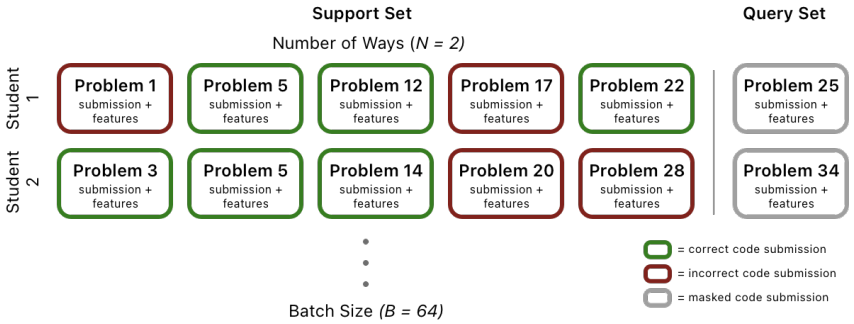

Figure 2: Diagram of meta-learning problem setup

3 METHODS

For our task, we employed the **Memory Augmented Neural Network (MANN)** architecture [2], which is a type of network equipped with external memory just like Neural Turing Machines. The model we created involved using a stacked sequence of LSTM's cells to encode the support examples into a dense vector which can then be used to make a predictions on the query example. We used two layers of stacked LSTMs for this model. Further, we created **hand-crafted submission features** with the aim of ensuring the model makes more informed predictions (described in the Appendix Section 7.1).

We also experimented with using the coding prompts themselves in our data representation to **encode information about Coding Concept based on Problem Prompt** in the model. The 2nd CSEDM Data Challenge organizers provide one-hot encodings of coding concepts required to correctly solve each problem. Some examples of the concepts represented in the encoding are: *for loop, boolean logic, if else, string equals, while loop, string concatenation, array indexing, and more* – in total, representing 18 possible concepts (not mutually-exclusive) for the prompt. We used these concept embeddings as input features to to help the model(s) learn what coding concepts students have learned that may carry over to future problems in order to improve future problem performance prediction. *Note that these concepts encodings are subjective judgement and not necessarily ground truth for coding problems.* With these coding concept encodings, we also tried training embeddings (8-dim output size) to represent these concepts instead of using the 18-dim one-hot encodings as direct input features. Since the one-hot encodings for prompt coding concepts had multiple concepts per prompt, we average the corresponding output embeddings. The motivation behind this was to provide dense orthogonal representation of each problem's required concepts to influence the model to identify the semantics of each coding concept.

In our baseline approach, we had manually crafted features given the submission history of a student. However, this was clearly sub-optimal as it is limited by our intuition of what might be useful features for this prediction problem. In this approach, we fed the sequence of submission history events into an LSTM for a given problem. We used embedding modules to automatically learn embeddings for discrete categories such as the *event type*, *compilation result* and *complier syntax error category*. We concatenated these dense embeddings vectors with each-other and made this the sequential input to the LSTM, the final output of which we treat as a **Sequential Submission History Embedding**. An overall diagram of this architecture is shown in Figure 5 in the Appendix (Section 7.2). Another powerful feature this architecture allows us access to is the *timestamp* between submissions. Specifically, for each submission event, we encode the timestamp offset (in seconds) relative to the first event in the sequence as part of the input vector. This allows the model to have information related to how long the student in question took in-between submissions to each problem, which can be a good indicator of whether they struggle with the concepts presented in that question.

Table 1: Accuracy and F_1 Score for Our Models

Model	Accuracy	F_1 score
Protonet with Manual Submission Features (Baseline 1)	85.4	87.1
MANN with Manual Submission Features (Baseline 2	86.3	87.5
MANN with Manual Submission Features + CC Embedding + SH Embedding	**90.5**	**90.9**

4 EXPERIMENTS

We will first describe our baseline models. We first implemented and ran a Protonet model, which is a type of Prototypical network that learns a metric space such that classification can be done by calculating distances to prototypes(representatives) for each class. Next, as described in the above section, we first implemented and ran the MANN model with manually-crafted submission features. Finally, we added submission history embeddings and problem concept embeddings to our MANN model (since the MANN model outperformed the Protonet model).

We describe the experimental details in the Appendix (Section 7.3) below. The main evaluation metrics we choose to use were *accuracy* and F_1 *score*. We include F_1 score due to the fact that our data is slightly imbalanced even after adjusting the sampling - approximately 58% of the query set has positive (correct) examples while only 42% has negative (incorrect) examples). Most of our experiments involved training a model using the approaches listed in the previous section, however, for approaches that seemed promising individually we did experiment with combining them together (ie. introducing two new types of features) to obtain the best results.

5 RESULTS AND ERROR ANALYSIS

Table 1 shown above summarizes our results of the approaches discussed in the Experiments section. In the table, CC Embedding refers to Coding Concept Embedding and SH Embedding refers to Submission History Embedding.

Now we will describe the important takeaways from our results:
Our final model, a Memory Augmented Neural Network (MANN) [2] architecture which concatenates submission history embeddings to handcrafted submission features and embeddings representing coding concepts (based on problem prompts), achieves an **accuracy of 90.5 percent and F1 score of 90.9 percent on this problem**. We now summarize our overall findings about the contributions of each of our approaches.

One of the most impactful approaches was adding *concept embeddings* to the model. This set of features individually raised by accuracy by around 3%. Intuitively, these features are useful as they provide information that allows the model to compare the relevance of the support set questions with the query set question. For example, the model can learn patterns such as recognizing if the concepts in the query set question also appeared in incorrect questions in the support set then it's likely the query set question would also be incorrect.

The submission history embedding provided a moderate boost in performance - around 1% in overall accuracy. This was aligned with our intuition since it provides the model with more flexibility to understand the user's submission history in addition to adding new temporal information that was not present in hand crafted features. Combining the Submission History Embedding with the concept embedding performs the best overall which again seems reasonable as these two techniques are relatively independent and thus amenable to being combined.

Our analysis of the errors that our model makes is shown below:
MANN Architecture: We inspected the kind of support and query sets in the test set which had the highest loss values. A common pattern we observed was that the model struggled with scenarios where the *first* problem in the support set had an incorrect label, the rest had positive labels and the query set problem had an incorrect label. This made intuitive sense to us which recurrent model generally struggle with longer temporal gaps in information for example, in this specific case, the

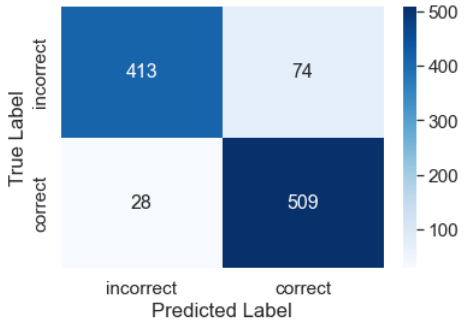

Figure 3: Confusion Matrix for MANN + Coding Concept Embedding + Submission Embedding Model

model would have to pass information about the first incorrect example through it's hidden state in order to use that in the query problem prediction.

Concept Type: Looking at the examples that the model gets incorrect, we investigated if there is any relationship between coding concepts in the support and query sets. We expected there to be significantly higher overlap in the coding concepts used between support and query sets for correct predictions compared to incorrect predictions, but we observed very minor difference (from average concept overlap of 63.8% in correct predictions to 62.7% in incorrect predictions). Thus, the remaining error is not attributable to lack of concept overlap between support and query sets.

Class Imbalance: In general, the model seems to perform slightly worse on meta-test examples where the query set problem is of the *incorrect* class. One explanation for this is negative examples appear less frequently in our dataset so the model has less opportunities to learn patterns corresponding to that specific class.

6 Conclusion/Future Work

In conclusion, we were able to successfully apply state-of-the-art meta-learning techniques to a novel dataset of student performance on coding questions from a public Computer Science course. Our findings were that some of the most impactful contributions were summarizing the submission history using a sequential embedding and adding information about the concepts related to each question. However, there are still ways to not only improve our model's results but also predict more information from the data. In particular, future work can incorporate multi-task learning approaches to try to additionally predict a student's final grade (or score out of 100) for the course, given performance on past problems. Further, in order to exploit the benefits of self-attention, future work should explore transformers in place of the LSTM model within the MANN architecture.

References

[1] ProtoTransformer: A Meta-Learning Approach to Providing Student Feedback https://arxiv.org/pdf/2107.14035.pdf Wu, Mike, et al. "ProtoTransformer: A Meta-Learning Approach to Providing Student Feedback." arXiv preprint arXiv:2107.14035 (2021).

[2] Dataset (CSEDM Data Challenge): https://sites.google.com/ncsu.edu/csedm-dc-2021/home.

[3] Code2Vec: https://arxiv.org/abs/1803.09473 Alon, Uri, et al. "code2vec: Learning distributed representations of code." Proceedings of the ACM on Programming Languages 3.POPL (2019): 1-29.

[4] CodeBERT: https://arxiv.org/pdf/2002.08155.pdf Feng, Zhangyin, et al. "Codebert: A pre-trained model for programming and natural languages." arXiv

[5] preprint arXiv:2002.08155 (2020). Pennington, J., Socher, R., Manning, C. D. (2014, October). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[6] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. 2015. Deep knowledge tracing. In Advances in neural information processing systems. 505–513.

[7] Dongmin Shin, Yugeun Shim, Hangyeol Yu, Seewoo Lee, Byungsoo Kim, Youngduck Choi. 2021. SAINT+: Integrating Temporal Features for EdNet Correctness Prediction.

7 APPENDIX

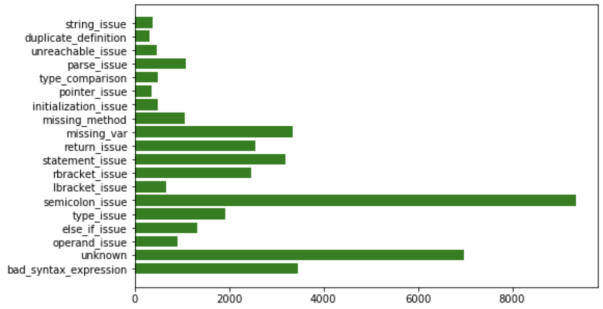

Figure 4: Histogram of 19 compiler error categories and their counts in the dataset

7.1 CREATION OF HAND-CRAFTED SUBMISSION FEATURES

A rich source of information is the compiler error message itself. This message can provide information about concepts the student struggles with (such as syntax, or not realizing they need to define a variable etc). To avoid the challenge that this information contains information specific to variable or method names, we manually grouped compilation errors into 19 major categories identified by key words in the error message output. By bucketing the error strings in this way, we were able to ensure each bucket had enough data which meant our model could easily use this information. To classify an error message into a bucket, we generally look for key phrases such as *already defined* (representing that a student doesn't understand they only need to define a variable once) or *initialization issue* (representing that a student doesn't understand they must initialize a variable before using it in Java).

7.2 SUBMISSION HISTORY EMBEDDING ARCHITECTURE

7.3 EXPERIMENT DETAILS

For all models, we used the *Adam* optimizer with an initial learning rate of 0.01. For the MANN architecture, we used two stacked layers of LSTMs with a hidden size of 32. The LSTM for the submission history encoding has a hidden size of 16, and the embedding layers for the event type and complier result have an output dimension of 5. Finally, we masked out all features except those related to the prompt (e.g. GloVe-based embeddings of the prompt and coding concept encodings needed for solving each prompt).

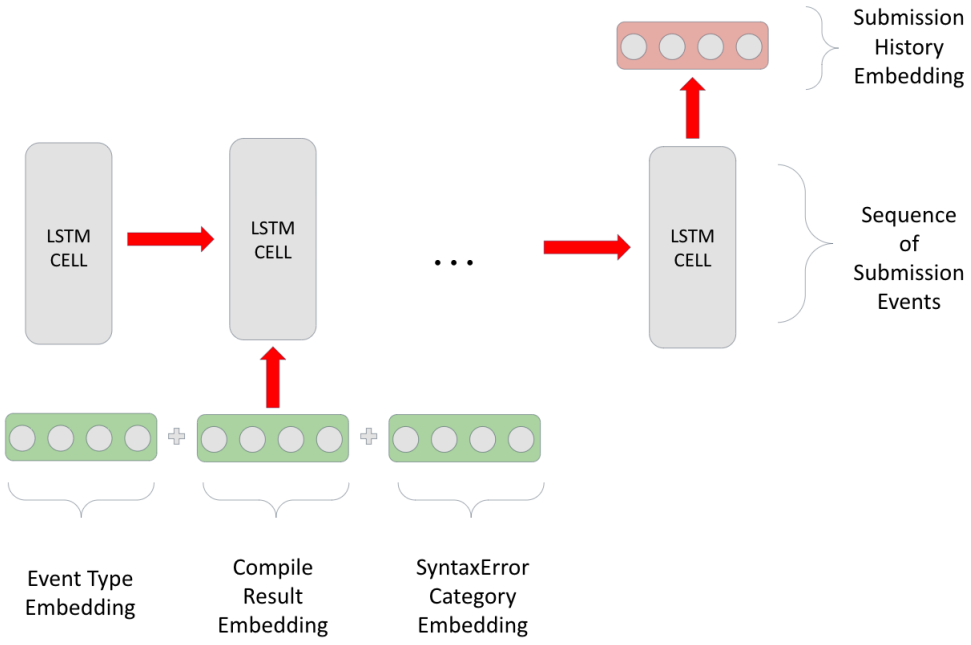

Figure 5: Architecture diagram of the Submission History Embedding generation. Information from each submission (event type, compile result, syntax error category) is embedded, concatenated and fed as a sequential input to an LSTM.