Tightly Coupled Worksheets and Homework Assignments for NLP

Laura Biester
Middlebury College
lbiester@middlebury.edu

Winston Wu
University of Hawai'i at Hilo
wswu@hawaii.edu

Abstract

In natural language processing courses, students often struggle to debug their code. In this paper, we present three homework assignments that are tightly coupled with in-class worksheets. The worksheets allow students to confirm their understanding of the algorithms on paper before trying to write code. Then, as students complete the coding portion of the assignments, the worksheets aid students in the debugging process as test cases for the code, allowing students to seamlessly compare their results to those from the correct execution of the algorithm.

1 Introduction

In natural language processing (NLP) and more broadly in machine learning (ML) courses, homework assignments frequently involve training a model that has been discussed in class on data provided by the instructor. Creating and training models is an important skill in NLP, but without proper scaffolding, such assignments can lead to open-ended questions posed to instructors and teaching assistants along the lines of "the accuracy of my model is lower than expected, but I don't know why or whether the current accuracy is acceptable." In part due to necessary implementation tricks (Xia, 2008) and the scale of data needed to train an effective model, NLP assignments often differ from those in other computer science (CS) classes, in which students can easily asses whether their solutions are correct or not.

This paper introduces an approach designed to mitigate this challenge—pairs of in-class worksheets and programming-based homework assignments using the worksheet examples as test cases—and then presents three such tightly coupled assignments created within this framework.

2 Development

The idea of tightly coupled worksheets and programming assignments stemmed from an NLP mini-course for high school students. The course took place over one week and was repeated three times over three weeks with new groups of students, allowing for rapid iterative improvement of the materials. The assignment was originally given with little scaffolding, and students struggled significantly to connect the exercise performed on a worksheet (sentiment classification with Naive Bayes using words as features) to the exercise performed in a programming lab assignment (language identification using character bigrams), even though the programming assignment included extensive starter code and significant real-time support (12-14 students programmed in pairs in a room with an instructor and a teaching assistant). In later iterations of the course, students were given a worksheet with a concrete example of Naive Bayes for language identification, and the assignment suggested that they use the same example as a test case in their code.

While this scaffolding was strictly necessary for high school students with minimal programming experience, we found that it can also be useful for undergraduates. In an NLP course for upper-level undergraduates,[1] the same assignment was used along with a similar tightly-coupled worksheet that students completed during class. The added scaffolding was particularly useful given that (a) students were primarily working on their programs without real-time instructor support and (b) the raw number of homework assignments for the course (seven assignments) was fairly high compared to other NLP courses, requiring that each homework assignment be slightly easier to complete.

[1]The course had data structures and discrete mathematics as prerequisites; prior experience with machine learning was not assumed.

Proceedings of the Sixth Workshop on Teaching NLP, pages 66–68

3 Approach

During the lecture, a worksheet is distributed to students. The worksheet's purpose is to provide students with an opportunity to practice the execution of the algorithm with step-by-step calculations on paper. This ensures that the student understands the algorithm before adding in the complexity of programming. Others have found similar worksheets helpful in reinforcing students' understanding (Eisner, 2002).

Then, a tightly coupled programming assignment asks the students to implement the algorithm in Python that they practiced with the worksheet. The student is provided two testing scripts: `test.py` and `test_mini.py`. `test.py` trains and tests the student's model on a large dataset and outputs accuracy or another metric as the final output. Meanwhile, `test_mini.py` trains and/or tests the model with the exact data that was provided on the worksheet. This allows students to easily see if their code's result matched the result from their worksheet, giving them an objective signal beyond standard ML metrics to help students determine whether their implementation was correct.

While students could in theory implement the same test cases that are written in `test_mini.py` independently, providing them to students streamlined the development process, allowing them to focus on the details of the algorithm. Using `test_mini.py` encourages students to develop good habits by testing with familiar data first when developing their own models outside of class.

4 Assignments

We describe three tightly coupled assignments, which are shared with this publication.[2] Complete starter code and autograders are available on request.

4.1 Assignment 1: Language Identification with Naive Bayes

The first assignment is to build a Naive Bayes language identification model with character n-gram features. While students eventually train and test their model on eight languages,[3] the worksheet focuses on just Spanish and English. Students count and smooth the character bigrams in three training instances, and then employ the model to classify a new word. The worksheet also includes a section on evaluating classifier performance by computing accuracy and creating a confusion matrix.

4.2 Assignment 2: Part-of-Speech Tagging with Hidden Markov Models

The second assignment is to build a Hidden Markov Model for Part-of-Speech tagging. On the worksheet, students fill out a table with the values that would be stored while executing the Viterbi algorithm using log probabilities. For homework, students write code to implement the Viterbi algorithm and optimize smoothing parameters.

4.3 Assignment 3: Beam Search for Text Generation

The third assignment is to implement beam search for text generation. The assignment assumes access to a model that outputs the probabilities of the n most probable tokens given a sequence of tokens.[4] On the worksheet, students perform beam search with the help of a Google Colab notebook that provides a high-level interface to interact with the language model (a function takes in token IDs as input and returns a dictionary where the keys are the n most probable token IDs and the values are their probabilities).[5] A similar function is provided as part of the programming assignment, and students' experience with it serves as a starting point when implementing the algorithm.

5 Student Reception

The instructor observed that most students relied heavily on the provided `test_mini.py` scripts and used them to debug during office hours. Students frequently referenced the calculations that they had made on the worksheets as part of their programming process. Some students needed to be prompted to check their model's internal data structures if the final result was incorrect, but by doing so were able to fix bugs in their code. Having ground-truth values of intermediate computations already worked out on their worksheets allowed them to isolate the component of their code that was not working. Afterward, multiple students gave positive unsolicited feedback about the structure of the assignments, such as they "liked how

[2] See this github repository.

[3] The languages included are Chinese (Mandarin), English, French, German, Italian, Russian, Spanish, and Turkish.

[4] We used GPT-2 (Radford et al., 2019), but another model could be plugged in here, or the assignment could be modified to focus on machine translation.

[5] All other worksheets could be completed on paper, although some students completed them on tablets with distributed PDFs.

the worksheets used the same examples that they started with for the programming assignments."

6 Limitations

6.1 Scope of On-Paper Test Cases

The examples on the worksheets provided to students did not capture all possible bugs that students could encounter while programming. While it would be possible to add complexity to the worksheets' examples to help students catch bugs early, there is some pedagogical value associated with allowing them to learn about how such features in the data could affect their models on their own.

6.2 Application to Pre-Trained Models

Applying this approach to large pre-trained models is less natural than applying it to Naive Bayes and Hidden Markov Models; for instance, it would be impossible to compute the correct output of a BERT model (Devlin et al., 2019) by hand on paper. Assignment 3 demonstrates one method for incorporating pre-trained models, but without any training. Future work could explore the possibility of creating small-scale test models for educational purposes that use the same API as large pre-trained models but have a minimal number of weights, which could then be used on worksheets and in `test_mini.py`-type programs.

Acknowledgments

The Hidden Markov Model for part-of-speech tagging assignment was adapted from an assignment created by Rada Mihalcea for an undergraduate NLP course at the University of Michigan.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jason Eisner. 2002. An interactive spreadsheet for teaching the forward-backward algorithm. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 10–18, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Technical Report*.

Fei Xia. 2008. The evolution of a statistical NLP course. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*, pages 45–53, Columbus, Ohio. Association for Computational Linguistics.