Learning How To Learn NLP: Developing Introductory Concepts Through Scaffolded Discovery

Alexandra Schofield
Harvey Mudd College
xanda@cs.hmc.edu

Richard Wicentowski
Swarthmore College
rwicent1@swarthmore.edu

Julie Medero
Harvey Mudd College
julie@cs.hmc.edu

Abstract

We present a scaffolded discovery learning approach to introducing concepts in a Natural Language Processing course aimed at computer science students at liberal arts institutions. We describe some of the objectives of this approach, as well as presenting specific ways that four of our discovery-based assignments combine specific natural language processing concepts with broader analytic skills. We argue this approach helps prepare students for many possible future paths involving both application and innovation of NLP technology by emphasizing experimental data navigation, experiment design, and awareness of the complexities and challenges of analysis.

1 Introduction

Discovery learning describes a pedagogical framing where, instead of introducing students to a skill and then using assessments to explicitly practice that skill, students are given a broad objective and "discover" pertinent concepts and skills in pursuit of that objective. Though pedagogical research yields unimpressive results for pure discovery learning (Mayer, 2004), several works support the effectiveness of *guided* discovery learning (Alfieri et al., 2011), where students are provided scaffolding for how to develop their solutions and regular opportunities for feedback or validation. This type of instructional approach offers the opportunity to exercise creativity and to take more ownership of their own sensemaking process.

In this paper, we present a proof of concept of the merits of scaffolded discovery learning by describing our implementation of four guided discovery learning exercises[1] to anchor the first four weeks of an undergraduate natural language processing course. We also detail why we have found this scaffolded discovery learning approach especially well-suited to an undergraduate NLP course.

2 Motivation

There are several strategic benefits to a discovery learning approach for NLP. First, the traditional computer science approach to NLP instruction tends to emphasize programming and/or core algorithms first (Bird, 2008), which requires students to jump immediately into implementing widely accepted algorithms for particular tasks. While this may lead to a more satisfying performance outcome, it may miss the opportunity for students to engage with why particular choices, such as how to tokenize text or how to smooth an n-gram model, actually make sense from a linguistic perspective. These skills are important when transferring to new domains, where the optimal choices might change and require further assessment.

Second, the choice of which model is the mostly "widely accepted" has been shown to change rapidly. Examining the curricular recommendations made by the ACM/IEEE Joint Task Force for Computing Curricula over time, we see speech recognition and parsing as emphasized topics in 2001, but no specific discussion of n-gram models (ACM/IEEE, 2001). In contrast, probabilistic models and ngrams appear in the 2013 version, while speech recognition disappears as an emphasized topic and parsing remains on the syllabus in vaguer terms (ACM/IEEE, 2013). While familiarity with these different problem spaces is helpful for building context and approaches to new problem domains, it is hard to predict if the depth of knowledge required to fully implement a conditional random field (CRF) or a deep neural net for machine translation will be useful beyond the next few years. In contrast, our discovery-learning labs focus on how to set up experimental data and protocols, select and interpret appropriate metrics, use those metrics to investigate what textual phe-

[1] Materials for these four assignments are available for use at https://github.com/DiscoverNLP.

Proceedings of the Fifth Workshop on Teaching NLP, pages 131–137

nomena they actually embody, and react to that knowledge with possible design interventions. We have found these skills not only generalize better across different models and problem domains, but also dominate the actual time spent doing our NLP work as researchers.

Third, this strategy of teaching is more compatible with the structure of the curricula at the undergraduate-only institutions where we teach. Upper-level courses such as natural language processing have shallow prerequisites and students have uneven backgrounds in math, computer science and linguistics. This challenge is shared by both computer science and linguistics undergraduate programs (Bird, 2008) and graduate courses aimed at an information studies audience (Hearst, 2005; Liddy and McCracken, 2005). Emphasizing skills of evaluation and model discovery over detailed model analyses helps ensure that students comfortable with data structures and some probability will be able to succeed and develop meaningful skills in the course. In our context, it has the additional benefit of inviting a liberal arts approach to critique the nature of what we evaluate in NLP tasks, as well as to consider the implications of the deployment of language technologies.

3 Learning Objectives

We present four labs as a central element of the first four weeks of a natural language course. In developing these exercises, we emphasize three skills we hope students develop through these assignments: analysis of quantitative results, text exploration, and generalizing concepts to new models. While some of these fall into the classic "text processing first" paradigm of course offered as an applied linguistics approach by Bird (2008), it does not do so at the exclusion of understanding the "why" of related algorithms and models. Notably, none of these skills specify particular models in natural language processing, though the labs we present do have some alignment with different classic NLP course topics. This is a deliberate alternative approach: by de-emphasizing outcomes focused on "knowing" particular algorithms and models, we provide more opportunity to practice skills to approach and interpret new models that students may choose to explore later in the course.

Analysis of quantitative results. Our work aims to support a progression not in terms of complexity of models, but in the levels of critique students apply to computational work on text. By building from some of the basic challenges of what it means to read, process, and write text in a computer through the development of increasingly complex metrics, students naturally have the opportunity to progress from making comments based on initially observed phenomena to making the case for whether results are surprising using typical metrics of the field. Unlike work that primarily privileges meeting certain minimum accuracy thresholds, these exercises intentionally include analyses of unimpressive results, helping to emphasize important lessons about how seemingly large quantitative differences don't always indicate significance in the vast, sparse universe of language.

Exploration of the text. Natural language processing benchmarks often present results in ways that obscure the contents of the data being evaluated, especially in cases where test data is totally hidden. Students new to the field of NLP often do not have strong intuitions around how much variation exists even in a limited problem domain. Examining system inputs and outputs helps them develop intuitions about language data and what it means to be "unusual" in a highly sparse space. Our discovery approach helps reinforce the importance of including both quantitative experimental results and qualitative discussions using examples about model behavior with respect to the text, as both contribute to an understanding of what a model has (or hasn't) done.

Thinking beyond individual models. An important component of practicing natural language processing is implementing models and comparing their performance. However, current "state of the art" models often are built using a combination of different fairly detailed neural architectural choices. Engaging with implementing these can require either many prerequisite courses, many days of in-class time to establish working knowledge of neural networks for sequential data, or a strategy of teaching how to code the models that eschews why to use these pieces. Further, with little confidence that these models won't be obsolete in two years, it can be an expensive investment to set up curricular materials for these topics without knowing whether they will still merit reuse the next time the course is offered. Because our approach de-emphasizes the specific models implemented in favor of understanding broader skills around evaluation and

comparison, we believe it will better prepare students to teach themselves and to ask meaningful research questions when approaching new work.

4 Course Format

The format of the two courses using these labs include both a lecture-focused larger class format and a smaller discussion-based format. In both, students are expected to read some text from the textbook, Jurafsky and Martin (2020), supplemented by papers on relevant NLP topics. Outside of course readings, the lab work constitutes the vast majority of the work completed outside of class for the first half of the semester.

In both formats, we reserve one day a week for specifically setting up and starting lab assignments. Students attend "lab day" in order to start work on the lab exercise for that week. The lab relies on the concepts discussed in class. Though the lab assignment is started during the lab period, the expectation is that the lab work will be completed outside of class and due the following week.

5 Lab Assignment Progression

The following subsections describe the progression of four laboratory assignments intended for the first four weeks of the course. Each lab assignment has a learning outcome related to analytical skills in natural language processing. Each of the four labs presented also focuses on introducing content from Jurafsky and Martin (2020).

These assignments are implemented in Python, with starter code provided through GitHub. The assignments can be coupled with autograders for the code managed through Gradescope.[2] When used, the autograders (which account for less than half the points in the student's final write-up) can serve as extra scaffolding, as they provide a useful on-demand tool for students to check whether their code is correct before writing their analysis.

Each lab is accompanied by an extensive lab write-up, which breaks the exploration topic into individual implementation goals, check-ins, and questions for students to address as they venture through the assignment. These are designed to be self-contained: while references to concepts or definitions described in the textbook reading may appear, most formulas and terms specific to the questions in the lab are re-introduced as part of the write-up. The text alternates between providing textual definitions and descriptions, coding instructions, and analysis questions. As the lab progresses, the questions grow more open-ended, culminating in prompts to make sense of results and what they signal about how well a model fits the text.

5.1 Lab 1: Regex Chatbot

In this lab, students develop a chatbot using regular expressions, either using Slack or Discord in our class offerings.[3] While the requirements of the assignment are fairly basic (for instance, the regular expressions must use groups and quantifiers), students are primarily graded on a write-up describing the chatbot and its performance. Students are expected to document with screenshots examples of conversations that worked well or poorly and to analyze what sorts of features might be missing.

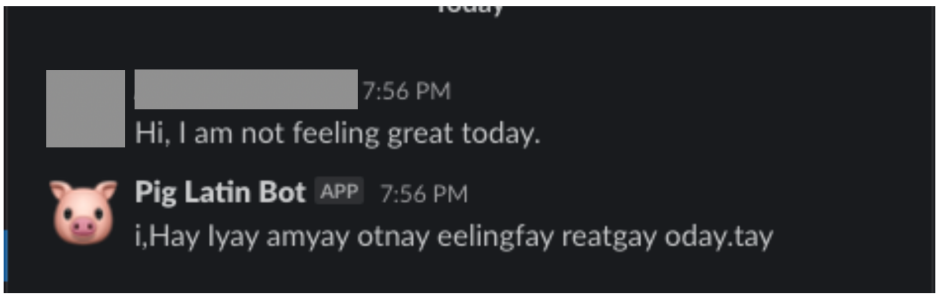
Preparation Students are expected to read the regular expressions subsection of the J&M textbook (§2.1). Students must also complete the first portion and at least one challenge puzzle from a regular expressions puzzle game called Regex Golf.[4].

Outcomes Having completed this assignment, students should be able to develop and test Python regular expressions, as well as to outline and illustrate a qualitative analysis of the capabilities and limitations of a new model. This assignment also encourages students to focus their time on thoughtful analysis instead of optimal performance prior introducing clear numerical metrics of performance. For instance, the sample of a student write-up in Figure 1 shows introspection related to a relatively simple pig Latin bot on how punctuation adds wrinkles to the program. These observations about what cases can lead to additional complexity help set up how we attend to variation in case, punctuation, Unicode, and more general human language variation in datasets for future labs. Though regular expressions are technically covered in another required course in the CS major as a theory topic, we find it important to solidify in an applied context, as tokenization is a necessary part of processing pipelines for the remainder of the course, a conclusion we share with Hearst (2005).

[2] https://www.gradescope.com/

[3] A side benefit of starting with this assignment is that, for courses that use one of these systems for Q&A, it encourages participation early among students.

[4] From Erling Ellinsen: https://alf.nu/RegexGolf



Here, however is a case where our SlackBot does poorly. It does not respond as intended, as we can see that it does not fix the capitalization of the first word. Moreover, the bot assumes that punctuation marks are a part of the word and includes them in the Pig Latin translation. If I had more time, I would be able to address this by having the regular expression ensure that the last character is a letter. Additionally, to fix the capitalization issue, I would need to check if the first letter of the word is capitalized, and if so, add its lower case version in the translation but then capitalize the first letter of the translated word.

Figure 1: A sample student write-up for Lab 1 describing a case where their bot failed to generate formulaic pig Latin and proposing improvements to address that case.

5.2 Lab 2: Tokenization and Segmentation

In this lab, students are first guided step-by-step through the basics of making a tokenizer in order to compute word frequencies in English texts from Project Gutenberg. Reflections include discussion of the effect of punctuation and lower-casing on which words appear most frequently. From there, students move on to improve a rule-based sentence segmenter for several portions of the Brown corpus using simple regular expressions. In this exercise, students report their precision, recall, and F1 scores, and reflect on which rules worked well, which did not, and why. Throughout both pieces, students explore the text, focusing on which text does not behave as expected.

Preparation Students are expected to prepare with textbook reading on text normalization (§2.2-2.4) as well as classification metrics (§4.7).

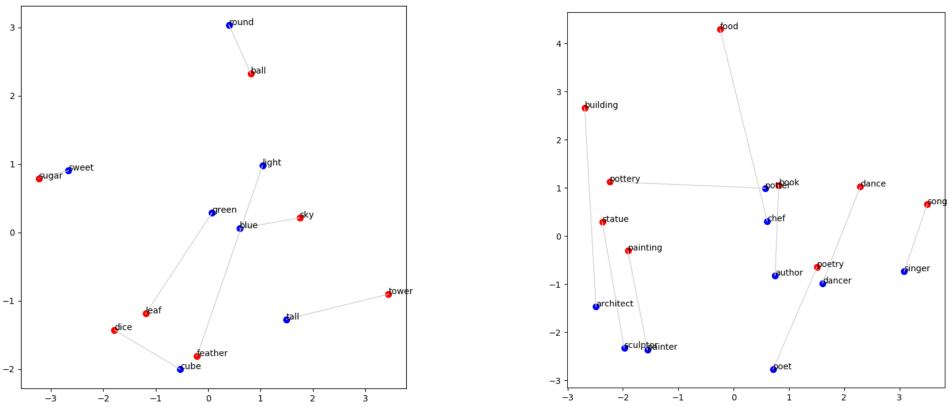
Outcomes Having completed this assignment, students should be able to perform standard string manipulations in Python. Students also will identify behaviors in text collections that fall outside typical prescriptive grammar rules, e.g. that sentences in the Brown corpus may start with a lowercase letter or end with a colon. Students are often tempted to add rules to their system for every sentence boundary, even those that are due to annotation errors. The analysis portion of this lab leads them to consider issues of overfitting for the first time, since adding rules for those edge cases leads to worse performance on held-out data. To complete the lab, students will need to combine these observations with quantitative results regarding word frequencies and classification metrics to describe what they have observed about their methods. Students will additionally reinforce Lab 1 skills of regular expression creation and identifying strengths and weakness in new systems.

5.3 Lab 3: Zipf's Law and N-gram Models

In this lab, students first look through the same Project Gutenberg texts to assess how closely the unigram frequency patterns match those projected by Zipf's law. Students then develop functions to extract unigram, bigram and trigram frequencies from the texts. They use these functions to compare features found in an unrelated data set. In the last offering of this course, we utilized the Hyperpartisan news dataset (Kiesel et al., 2019) in order to understand how many types and tokens are unique to hyperpartisan or non-hyperpartisan-labeled news. Students explored these same evaluations on a random reshuffling of the data to understand the magnitude of "unique" attributes that can arise even when comparing texts from the same source.

Preparation Students are expected to prepare with textbook reading on n-gram language models (§3.1-3.4).

Outcomes Having completed this assignment, students should be able to develop code to extract n-gram frequencies from text and predict unigram frequency distributions from Zipf's law. This requires interpreting relative and absolute frequencies and their reasonable values: a common error



(a) Properties of objects

(b) Creators and creations

Figure 2: Two examples of alternate analogy tasks developed by students for Lab 4.

scenario as students attempt to compare empirical versus theoretical behavior is to plot absolute theoretical word frequencies against relative empirical word frequencies, as displayed in Figure 3. Further, they should be able to provide qualitative and quantitative analysis of rare and common features and their effect on different frequency statistics.

The datasets used for this lab are too large for naïve file-reading solutions that load the whole file into memory at once. Students learn to iteratively process the data, which prepares them for working with large files in later labs and course projects. As they work with these files, students gain some practice leveraging existing libraries (such as `lxml`[5], `spaCy` (Honnibal et al., 2020), and `matplotlib` (Hunter, 2007)) to process text and plot data.

5.4 Lab 4: Word Embeddings

In this lab, students use GloVe embeddings (Pennington et al., 2014) to experiment with relationships between word vectors, including word similarities and word analogies. Students first practice loading in text vectors and saving them as a compressed `numpy` matrix. Subsequently, they develop code to rank similar words for a selection of related words, as well as to test whether certain relations (e.g. gender, number, comparative vs superlative) have consistent vector differences as rendered in 2d plots using PCA. Finally, students develop their own relations and word lists to test the consistency of this method.

Preparation Students are expected to prepare with textbook reading on vectors space models (§6.1-6.5) and their evaluation (§6.10-6.12). Section 6.8 on word2vec or an introduction to GloVe may boost confidence.

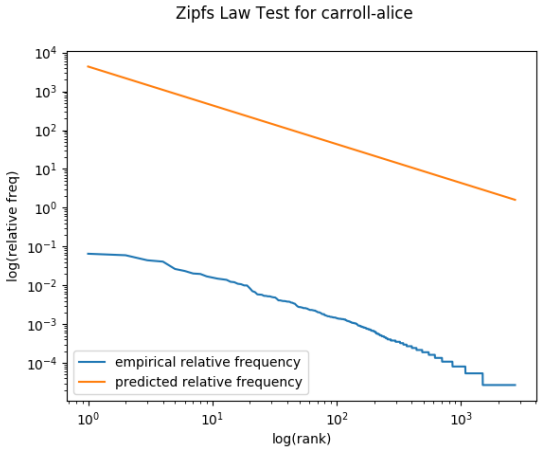


Figure 3: An incorrect plot for Lab 3, caused by a student conflating relative and absolute frequencies. Autograders and lab check-in points allow students to engage with these points of confusion while still catching errors before students write their analyses.

Outcomes Having completed this assignment, students should be able to perform mathematical operations common to vector space models, such as computing cosine similarity, vector norms, and average vector differences. Students will know how to use `numpy` operations to speed up computation (Harris et al., 2020) and dimensionality reduction to visualize higher-dimensional behavior. Additionally, students should know how to develop and test a hypothesis about word vector geometries using appropriate metrics, visualizations, and qualitative insights. These hypotheses can lead to broader understanding of what types of relationships work well as word embedding analogies, too, including

[5] `https://lxml.de/`

how relationships that are not one-to-one may fail (Figure 2a) and how polysemy adds wrinkles to word analogies (Figure 2b).

6 Observations

In many ways, this format has helped us to feel liberated in how we approach topics later in the semester. In the most recent offering of the course, for instance, the latter half focuses on student projects and presentations on student-determined topics covering a range of popular tasks like machine translation and question answering and more wide-ranging topics like text-to-speech systems and computational social science. In other semesters, the course has continued with structured weekly (or bi-weekly) lab assignments, often culminating in a lab assignment that introduces a shared task that students then work on as a final project. Overall, we find that the foundational skills introduced in the first four weeks prepare students well for all of these paths through the rest of the course.

This approach admittedly releases the opportunity to do a deeper technical dive with the class into modern models; while students tend to discuss LSTMs, BERT, and other popular modern models, they are not expected to fully present these topics in their presentations. While we had initially worried about whether this would provide too little infrastructure to make sense of neural models, we have been happy with how students use the skills we emphasized earlier to uncover known controversies in the field of NLP. We have witnessed advanced discussions initiated by the students about topics such as how strong baselines can be in question-answering datasets, the possible shortcomings of metrics such as BLEU, ROUGE, and PYRAMID, and what it would mean to reach human parity for machine translation.

Taking a discovery approach is not without risks, especially when students may get stuck or confused. We have learned some lessons about where more guidance may be needed for students. First, unsurprisingly, file I/O often can be remarkably picky, and we have found that the I/O portions of these labs can be fairly brittle when students make choices that may have been reasonable in previous classes with fairly rudimentary file processing. Further illustration around ways to load and save data, manipulate `numpy` and `spaCy` objects, and otherwise use libraries can greatly reduce time students spend stuck on smaller bugs instead of NLP questions. Additionally, students often ask a number of questions about how much analysis is sufficient and which aspects to focus on, often not finding engagement with the text intuitive. Sample write-ups from the first lab or two may help illustrate this, as well as more development on the prompts for the write-up to elicit thoughtful analyses.

7 Conclusions

We hope that the description of this class format and assignment progression helps motivate the feasibility of a discovery-based approach to teaching natural language processing. We are delighted to share the initial assignment instructions, lab files, and student Gradescope image publicly on Github at `https://github.com/DiscoverNLP`, with additional files for autograding through Gradescope available on request.

8 Acknowledgements

We would like to thank the students at both Swarthmore College and Harvey Mudd College who participated in and gave feedback to this class, including the students who gave permission for their work to be shared in this paper.

References

ACM/IEEE Joint Task Force on Computing Curricula. 2001. Computing curricula 2001. `https://www.acm.org/education/curricula-recommendations`.

ACM/IEEE Joint Task Force on Computing Curricula. 2013. Computing curricula 2013. `https://www.acm.org/education/curricula-recommendations`.

Louis Alfieri, Patricia J Brooks, Naomi J Aldrich, and Harriet R Tenenbaum. 2011. Does discovery-based instruction enhance learning? *Journal of Educational Psychology*, 103(1):1.

Steven Bird. 2008. Defining a core body of knowledge for the introductory computational linguistics curriculum. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*, pages 27–35, Columbus, Ohio. Association for Computational Linguistics.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del

Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362.

Marti Hearst. 2005. Teaching applied natural language processing: Triumphs and tribulations. In *Proceedings of the Second ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, pages 1–8, Ann Arbor, Michigan. Association for Computational Linguistics.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. *spaCy: Industrial-strength Natural Language Processing in Python*. Zenodo.

J. D. Hunter. 2007. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Dan Jurafsky and James H Martin. 2020. *Speech and Language Processing*, 3rd (draft) edition. Prentice Hall.

Johannes Kiesel, Maria Mestre, Rishabh Shukla, Emmanuel Vincent, Payam Adineh, David Corney, Benno Stein, and Martin Potthast. 2019. SemEval-2019 task 4: Hyperpartisan news detection. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 829–839, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Elizabeth Liddy and Nancy McCracken. 2005. Hands-on NLP for an interdisciplinary audience. In *Proceedings of the Second ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, pages 62–68, Ann Arbor, Michigan. Association for Computational Linguistics.

Richard E Mayer. 2004. Should there be a three-strikes rule against pure discovery learning? *American Psychologist*, 59(1):14.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.