When I began teaching computer science through tutoring and programming workshops, I regularly was disappointed with my lack of efficiency. Unlike the TAs and professors I looked up to, I could not instantly diagnose my students’ programming errors. I felt I moved too slowly: I would ask lots of questions, admit I was confused too, and then guide them through the steps I would take to solve the problem myself. However, in my seventh semester of undergraduate TAing, a classmate told me that when I had been her TA, my strategy of one-on-one teaching—showing my uncertainty and my process—was part of what convinced her to stay in a computer science degree in the face of severe imposter syndrome. Her experience helped me both appreciate and refine my empathy-driven approach to supporting all students as they tackle the many challenges of rigorous computer science curriculum.

Since then, I have spent my graduate career seeking opportunities to educate and empower students as technology creators. I co-designed and co-taught Text Mining for History and Literature with David Mimno, in which we used a flipped classroom model to teach applied text mining concepts to our class of 30 people. In my current role as a co-instructor alongside Éva Tardos for Introduction to Analysis of Algorithms, a core lecture course with 300 students, I have further developed my ideas of how to design curriculum, assessments, lectures, and resources that can provide a compassionate learning environment for large courses. I have also taken on a variety of roles as a TA and substitute lecturer, all the while supplementing my classroom experience with more formal training, including taking a course on Teaching in Higher Education offered by Cornell’s Center for Teaching Excellence. Through this training and experience, I developed a teaching strategy that centers three things for my students: 

opportunities to communicate, 
a comfortable space to make mistakes, and compassion.

I find that these pillars can help all students succeed, especially those who carry the additional burdens of impostor syndrome or stereotype threat.

Communication. Perhaps the most important skill I teach to my students is how to communicate technical material. Without effective communication, research cannot build on other research, engineers cannot agree on designs, and users cannot learn what computation can do for them. The strategy of “rubber-duck debugging” also shows the value that communicating about our work can have: if a student can clearly articulate a problem they have encountered, that process alone may help illuminate the solution.

Like any skill, communication requires practice and feedback to hone, which I like to encourage through active learning. In the Text Mining course I co-taught, we used a flipped classroom to maximize the amount of interaction and discussion that took place in class: students practiced programming during class on Mondays and Wednesdays and arrived ready to discuss readings on Fridays. While sometimes this reduced how much material we could include in a given class, students were able to explore more deeply and better appreciate that material, and thus better retained the lessons from these in-class programming assignments and discussions. Not only did students’ conversations about programs and readings help to solidify their understanding of the material, they also provided a comfortable entry point to explore these topics with people outside of the class. The course led several humanities Ph.D. students and computer science undergraduates to continue to pursue computational text analysis projects after the course ended, and has had growing enrollment with successive offerings.

Algorithms takes on a more traditional class structure, currently built around three 50-minute lectures a week. We use iClicker questions to both gauge understanding and encourage students to talk to their neighbors about concepts. When I ask a two-minute question about what shape of table might work to solve a dynamic programming problem, students get the chance to test their communication of problems with each other before deciding on an answer themselves. These questions also lure students out of the false sense of comfort with the material that passive, lecture-based learning can provide, into active application of the concepts in lecture to a new setting.

Room to make mistakes. Edgar Dijkstra said, “If debugging is the process of removing bugs, then programming must be the process of putting them in.” I aim to teach students to recognize and debug errors in code and theory alike by making a comfortable space for students to engage in the process of creating and finding their errors. In Text Mining, my advisor and I would often introduce (or re-introduce) bugs
in our code to provide short examples in class of how to debug and, equally importantly, to reinforce how normal bugs are in programming. In *Algorithms*, my responses to incorrect student answers in class aim to bring out the good idea or interesting problem they noticed when forming their answer. For instance, in a recent lecture, while helping students review which way to perform reductions for NP-completeness, I took the opportunity to explain several reasons why people might understandably find it counterintuitive to reduce *from* the known problem, but confirmed that this process is actually correct. I always encourage them to help me catch notation mistakes and unexplained cases in my notes, and when time permits, I show my process of fixing those errors in class.

Outside of class time, I have cast myself as a safety net for students trying for ambitious projects. In my time as a TA, a mentor for undergraduate independent studies, and (before graduate school) a mentor of two interns, I perceived my role as a guide: I would help to build and refine ideas, then lightly encourage their progress without managing too closely. Initially, I often interpreted silence from students whose projects I advised as a good sign, that they were progressing without my assistance, when actually it was often an indicator of being stuck and too embarrassed to ask for help. Now, in supporting my two undergraduate mentees, I’ve established a new policy that they should always feel comfortable emailing me if they get stuck. Sometimes I might just encourage them to press on, while other times I will use it as an opportunity to introduce new concepts or review complicated ones to help them along their way. I supplement my work as a guide small amounts of explicit structure for accountability, such as regular meetings and reports. My current students have been making progress quickly and with more enthusiasm than ever; one of my current students expanded the number of credit hours on her project because it was her favorite work this semester.

**Compassion.** Engineering students in strong undergraduate programs are placed under a great deal of stress to be at the top of a class of other exceptional people. While I find graded evaluations and rigid course policies to be an important part of meaningful evaluation of students, I try to also respect the time and individual lives of my students when structuring course evaluations. When possible, I balance more traditional assignments with effort- and participation-based small assessments of understanding, such as mini-quizzes, clicker questions, and online participation. In my algorithms course, I prefer to give points on homework assignments based on a student’s process: for instance, I often give more than half of the possible credit if they use the appropriate structure to attempt a proof of an incorrect algorithm.

In *Text Mining*, a number of the students had written no code before the course started. Effort-based grading of in-class programming assignments gave students the space to experiment with new things, as well as direct access to instructors while working on the assignments to help ensure they could learn from errors. It also required proactively designing structures to allow students to govern their own time, such as accepting rewrites for some assignments, providing late days or quiz skip days, and accepting reasonable student requests for extensions. This sort of flexibility benefits all students and allows them to be more in control of their schedules and accountable for their work, but it also particularly helps students who might feel excluded or uncomfortable in the traditional formats of discussion, assignment work, and question-answering often found in computer science courses.

**Future Goals.** As an instructor, my teaching has ranged from very close to my research, as in *Text Mining*, to very distant, as in *Algorithms*. I have a broad background in computer science and education experience, and am capable of teaching core computer science courses as well as upper-level courses and electives in machine learning, natural language processing, and theory.

If I were to continue teaching Algorithms, I would love to develop more ways for students to use project and group work to build their interest in algorithmic applications. As an NLP researcher, I would also like to provide an upper-level, project-centered NLP course to give students the chance to develop workshop-quality research. Finally, I am interested in integrating my research and previous study associated with English Literature into teaching a first-year writing course focused on ethics in artificial intelligence and its perceptions in popular culture. This course could provide an interesting basis to explore complicated questions that could invite STEAM students to better engage with their future technical work in a more holistic way while simultaneously teaching the core communication skills so essential to that work.
Assessments for CS 4820: Analysis of Algorithms

<table>
<thead>
<tr>
<th>Question (1 = strongly disagree, 5 = strongly agree)</th>
<th>Average score</th>
</tr>
</thead>
<tbody>
<tr>
<td>The instructors lectures are clear. (Includes notes, speaking voice, structure, etc.)</td>
<td>4.43</td>
</tr>
<tr>
<td>The instructor is knowledgeable about the course material.</td>
<td>4.72</td>
</tr>
<tr>
<td>The instructor is approachable and responsive outside of class time.</td>
<td>4.42</td>
</tr>
<tr>
<td>The instructor is helpful in one-on-one interaction through office hours or online.</td>
<td>4.24</td>
</tr>
<tr>
<td>Compared to other instructors at Cornell, this instructor is an effective teacher.</td>
<td>4.39</td>
</tr>
<tr>
<td>The instructor has stimulated my interest in the course subject.</td>
<td>4.19</td>
</tr>
</tbody>
</table>

Quotes (full forms available on request)

- Love how relatable she is, and very talented lecturer! Makes things sound very real-world and simple, though they are complicated.

- I love Professor Schofield! She is extremely clear and seems really passionate about the subject material. I also find her really relatable and an enjoyable professor who brings humor and levity to the classroom, making a class that I find difficult and not intuitive enjoyable.

- Incredibly clear lecture notes, very clear introduction into the algorithms, very good pace, good at explaining the “why,” and I’m a big fan of the rainbow pen. Professor Schofield brings a level of energy and verve to a 9 am lecture that is unmatched and infectious.

- Prof Schofield is incredibly passionate about the subject matter taught and has very actively tried to design and facilitate a class that allows students to grasp the material with more focus on rigor than many other undergraduate CS courses.

- By far one of the best professors I’ve had, even outside of Computer Science. Her notes and lecturing are very clear and understandable, and I really appreciate the effort she makes to make the course topics as relatable as possible by using examples relevant to our lives as students. Her perky attitude always helps get through the early 9am lecture.

- (What do you like about the course?) The level of effort the professors place into the course. Things like student-professor lunches show that they truly care about the students. This ensures me that I can go to the professors and ask about the course material or just about anything else I’d like help/advice on.

- (What do you like about the course?) I honestly really enjoy how challenging it is. I feel like I’m learning a lot of very useful algorithmic knowledge that will be helpful in my career.

- Algorithms feels like the best that the field of computer science has to offer, taking it with Professor Tardos and Professor Schofield. They make the problems feel significant and the solutions elegant and understandable. Having six late days is great. The textbook is really clear and well-written in my opinion. The people on piazza are mostly helpful, and I like that the professors answer the question much of the time.

---

1 Full forms are available on request. 280 students responded to this survey. Due to Cornell’s TA assessment structure, courses prior to this appointment included only one or two full student evaluations.