Python as a Vehicle for Teaching Natural Language Processing

Reva Freedman

Northern Illinois University
Department of Computer Science
DeKalb, IL 60115
rfreedman@niu.edu

Abstract

I have taught Introduction to Natural Language Processing several times at Northern Illinois University. Since the students are more interested in the Python code that holds the assignments together than in the NLP content, over time I have cut back on linguistic content and added to the programming content. This year I renamed the course "Introduction to NLP in Python" and spent the first few weeks teaching Python from scratch. This decision has been very successful. The syllabus interweaves Python topics, core NLP topics, and essential computer science topics I feel every student should learn before graduating. In this paper I describe the motivation for teaching the course this way, the syllabus and its rationale, and prospects for expanding the course to two semesters.

Introduction

I have taught Introduction to Natural Language Processing several times at Northern Illinois University. Since the students are more interested in the Python code that holds the assignments together than in the NLP content, over time I have cut back on linguistic content and added to the programming content. This year I renamed the course "Introduction to NLP in Python" and spent the first few weeks teaching Python from scratch. This decision has been very successful.

Although I had to remove some linguistic content to make this change, it was all theoretical material: I did not have to remove any material that was needed for a programming assignment. Earlier versions of this course that were more linguistics-oriented are described in Freedman (2005, 2008).

The syllabus interweaves Python topics, core NLP topics, and essential computer science topics I feel every student should learn before graduating. In this paper I describe the motivation for teaching the course this way, the syllabus and its rationale, and prospects for expanding the course to two semesters.

I chose Python because I like Python, because there is excellent pedagogical software for teaching natural language processing in Python with an accompanying textbook (NLTK; Bird, Klein, and Loper 2009), and because we didn't already have a course in Python. Additional advantages for programming in Python are that it gives us another chance to teach good programming style, and it enables us to give the students a tiny taste of functional programming, which should also improve their programming style and throughput.

In this paper I describe my experience teaching NLP to students with no previous background and the syllabus I have developed as a result. An overview of the syllabus is shown in Figure 1.

Student background

Northern Illinois University is a large state university located about 60 miles west of Chicago. NIU has a B.S. program in computer science and a coursework oriented M.S. Most undergraduate majors come from the suburbs of Chicago or from small towns near the university. Some of the graduate students come from a similar background, but most are international students.

For the majority of students, undergraduate and graduate, the preferred career path is generally to obtain a programming job in local industry, preferably in a hi-tech area. Most undergraduates who take the course do so out of a desire to do something different from their required courses. Many of the graduate students are looking for a course that does not require the prerequisites of their core courses. The possibility of learning a new computer language that might help them in the job market is a strong draw. Some graduate students are also interested in improving their programming skill by starting over in another language.

The following background information has influenced the design of the course. The information is derived from talking to prospective and actual students as well as from several years of anonymous course evaluation forms from Introduction to Artificial Intelligence and Introduction to NLP.

1. *Motivation for taking the course.* Students are taking the course for fun or to learn Python. They are unlikely to need specific NLP content in their future careers.

2. *Linguistic background.* Students have never studied linguistics and are not particularly interested in it. In general, they are not familiar with the common syntactic constructions of English taught in traditional English grammar and are often unsure about parts of speech, even though most have studied a foreign language in high school.

3. *Academic sophistication.* Students are used to getting information from the web and are uncomfortable having to use offline reference materials. They are not comfortable with or interested in traditional academic prose or research papers. They expect to have everything they need for each assignment explicitly taught in class.

4. *Programming ability.* Students are not familiar with programming languages other than conventional imperative languages such as C++, Java, and .NET. They have a wide range of programming ability, where the best have professional-level skills and the weakest should probably not be in a graduate class.

Course goals

The course has three goals:

1. Give students a general background in the issues involved in handling written text, some of the most common applications, and some of the most widely used algorithms.
2. Provide students with a productive experience in a modern programming language.
3. Teach students a number of useful concepts that they might not otherwise come across in their course of study. These topics include:

- Bayes' Law (e.g., spelling correction)
- Dynamic programming (e.g., chart parsing)
- Regular expressions and finite-state machines
- Context-free grammars
- Text mining via rule induction (C4.5)

The following sections of the paper describe the most important units of the course, showing how they use the principles stated above to contribute to these goals.

Python

Basic Python

I teach Python from scratch for two reasons. For the stronger students, I want them to have an opportunity to learn a new programming style from scratch. For the weaker students, I want to give them the opportunity to catch up. As computer science majors, the students tend to find that the treatment of Python in the NLTK textbook does not answer enough of their technical questions, such as issues on argument handling and copying of objects vs. references to them.

I give several lectures on Python, including the following topics:

- Basic data structures
- Basic control structures
- Functions
- File handling
- Associative arrays
- Options for implementing trees
- Basic functional programming
- Objects

I also give several lectures on Python libraries, including the following topics:
- GUI programming with Tkinter
- Reading and parsing web pages
- Regular expressions

The online Python tutorial (http://docs.python.org/tutorial) is the main reference for the topics in the first group. I also use the Python tutorial for GUI programming. I use the Dive into Python web site (http://www.diveintopython.org) for regular expressions and web programming. I have not found a good reference for teaching about objects in Python or for the Python memory model.

Although it is easy to find references for the functional programming constructs themselves, I have not found a good reference for motivating students to use these constructs or for helping them learn to visualize their programs in terms of these new constructs.

This section of the course basically covers the material in chapters 1–4 of the NLTK textbook. In the end students cope well with a language that allows them to code much faster than C++ with less debugging. They all learn to use Python lists and associative arrays (dictionaries). Most learn to use some of the functional programming capabilities available, such as list comprehensions, although not to the degree I would like.

Allowing for student differences

To prevent the best students from getting bored, I allow them to use any construct or library they like, whether I have taught it in class or not. I also permit them to use other GUIs, as some feel strongly that wxPython is superior to Tkinter.

Week 1: Program 1 – Counting letters and words; using the compiler and editor
Week 2: Homework 1 – Simple functional programming practice
Week 3: Program 2 – Count by first letter; use of associative array (Python dictionaries) and data files
Week 4: Program 3 – Web programming with `urllib`; functions and objects
Week 5: Program 4 – Interactive Eliza-like system; regular expressions
Week 6: Program 5 – GUI front-end to an earlier assignment; Tkinter
Week 7: Program 6 – Part-of-speech tagging; serialization via the *pickle* module
** First exam

Week 8: Program 7 – Email spam classifier; text mining with C4.5, regular expressions to identify features
Week 9: Program 8 – Parsing with finite state machines
Week 10: Program 9 – Parsing with context-free grammars; implementations of trees
** Choose project topics
Week 11: Homework 2 – Earley's chart parsing algorithm; dynamic programming (paper/pencil)
Week 12: Program 10 - Information retrieval with tf–idf
Week 13: Homework 3 - Spelling correction via Bayesian statistics (paper/pencil)
** Second exam

Week 14: Slack – could be used for latent semantic analysis or another theoretical topic
Week 15: Project presentations

Figure 1: Outline of syllabus

Later on in the course, I let them use the data mining algorithm of their choice. Many prefer support vector machines (SVM, Vapnik 1995) because they have learned it in our pattern recognition class. I prefer C4.5 because it produces a decision tree rather than just a binary decision, and because I would like to show how machine learning can be accomplished with only elementary mathematics.

Use of Python libraries

The course includes several topics based on well-established existing libraries. These topics include simple web programming, regular expressions and GUI programming.

I teach simple web programming, e.g., how to access a web page from a program. First, it's useful and it impresses students as to how easy it is to do in Python. Second, more advanced usage, such as HTML or XML parsing, gives them an opportunity to practice functions and objects. Finally, accessing web content from a program is a skill highly desired by students.

I teach regular expressions for two reasons. In addition to being required for continued use of the NLTK textbook, regular expressions are an important idea that students do not necessarily encounter in another programming class. We experiment with Rocky Ross' interactive web site (Pascoe 2005) and a web site where students can build their own tables at http://scoter3.union.edu/~hannayd/csc350/simulators/FiniteStateMachine/fsm.htm.

As an example of an NLP application, albeit a shallow one, that can be implemented with regular expressions, we experiment in class with Eliza, trying both to make it work and make it fail. I give out a list of versions available on the web, and students can easily find more. In class I often use the emacs built-in version. I then give out copies of the original Eliza paper (Weizenbaum 1966), which contains the original script in an appendix, although the coding style in that paper is not useful for students to emulate.

The first major project in the course is to write their own Eliza-type program. Students choose a realistic but limited domain such as a database front-end. This project is implemented in Python as a rule-based system with heavy use of regular expressions. If the class is small enough, I have students do a short presentation of their domain, including a sample conversation, before they write their code. After the projects are due, they present their results to the class.

Finally, I teach Tkinter because it's easy to learn and students benefit greatly, both for class projects and elsewhere, by knowing a GUI framework. Some of the better students prefer wxPython.

NLP

This section of the course covers key sections of chapters 5, 6 and 8 of the NLTK textbook.

Part-of-speech tagging

I follow the treatment in the NLTK textbook, which starts from a simple baseline and adds features to improve accuracy. Although most of the students have no intrinsic interest in parts of speech, the more adventuresome ones appreciate this unit because they enjoy tweaking the features and increasing the size of the training set to improve the accuracy of the algorithm. I confine my comments in class to the common parts of speech that they already know. An important principle students learn during this unit is that accuracy is strongly affected by the quantity of training data used.

I also teach the use of Pickle to create persistent objects at this point because a tagger good enough to handle the test data is quite large.

Text mining

Next I present a simple approach to text mining using C4.5 (Quinlan 1993). The project for this unit involves identifying spam in a corpus provided by the makers of SpamAssassin (http://spamassassin.apache.org/public corpus/). Students enjoy guessing features in email headers and text that might indicate spam. They write regular expressions to identify these features, then use the data acquired as input to the classifier.

Finite state machines

Finite state machines are another topic that I believe students should not graduate without being exposed to. My linguistically naive students are more willing than most linguists to believe that FSMs can be used to recognize English. It is interesting to note that their FSMs are quite different from those derived by linguists. From a Python point of view, FSMs are another application of the dictionary data type, which is one of Python's most versatile creations.

Context-free grammars and parsing

I present several approaches to parsing, since many students will not otherwise see context-free grammars in their studies. These include the top-down recursive descent parser, the bottom-up parser, and the chart parser described in the NLTK book. NLTK has some beautiful demos that show these parsers running step by step. These demos are extremely helpful in showing students how the parsers work.

The assignment for this unit involves writing a small CFG to handle the same sentences as the FSM assignment. By this time students have acquired some fluency with the common parts of speech, although their CFGs are again quite different from those a linguist would develop. NLTK returns the parses as parenthesized trees, which means that this is a good time to demonstrate different ways to implement trees in Python, as Python does not have a primitive tree data type.

Since the reaction of a previous class to Earley's algorithm was "we understand it; it's just not interesting," I frame Earley's algorithm as an example of dynamic programming, again a topic that I feel students should not graduate without knowing.

Information retrieval using tf-idf

As an example of a numerical technique I teach information retrieval using tf-idf (Salton 1988). This year I used each of the 70,000 sentences in the Brown corpus as a "document." Students were impressed by the behavior of such a simple formula. The Porter stemming algorithm (Porter 1980) is an ugly but practical way to remove prefixes and suffixes from words. I teach it at this point so that students can try information retrieval with and without stemming, a well-known issue in information retrieval. I use the treatment of it in Jurafsky and Martin (2009).

Spelling correction via Bayes' Law

I present Kernighan, Church and Gale's (1990) Bayesian approach to spelling correction, as explained by Jurafsky and Martin (2009, section 5.9).

Kernighan et al. choose the correction that maximizes $P(t|c)P(c)$, where t is the typo and c is a candidate correction. I teach briefly about Bayes' Law in general, as I feel it is a topic every computer scientist should have in their toolbox. Many students have studied Bayes' Law in a statistics class, but have not seen it used in this fashion. I motivate Kernighan's formula by showing a picture of the veery, a small brown bird. Although duplicating a letter is a common typo, "veery" is an uncommon word and is unlikely to be the one intended by the average student.

Students choose a corpus and replicate Kernighan's calculations. They then compare their results to results from their favorite word processor. They are generally surprised at how similar the results are from what they originally see as an unmotivated calculation. They are always surprised to learn that spelling correction is generally not done by a lookup process. They are also surprised to learn that results are largely independent of the corpus chosen.

I also demonstrate approximating word frequencies by page counts in Google, along with a discussion of the advantages and disadvantages of doing so. In general, students prefer to obtain word frequencies from one of corpora included with NLTK or a similar corpus downloaded from the web.

Course project

Once most of the programming assignments for the course are complete, I introduce the course project. To give students time to work on it, I save some of the paper-and-pencil simulations for the end of the semester.

I give students a list of several possible projects, including an interactive conversation system or game, a text mining project, or extending the part-of-speech tagger, FSM or CFG they have already built. Students can also develop their own project. I encourage international students to redo the part-of-speech tagger, FSM or CFG assignment in their native language; however, most students prefer a programming project or an extension of the text mining assignment. For weak programmers, I provide some non-programming projects such as evaluating a tagger or evaluating one of the large probabilistic parsers currently available.

Future work

Now that the concept of teaching NLP by carefully sequenced Python programs has been established, I would like to extend the course to a second semester. Python topics requested by students include a more detailed treatment of modules, objects and the memory model, and additional web programming.

I would like to include several more fundamental computer science topics in the second semester. Instead of utilizing the top-down and bottom-up parsing algorithms included in NLTK, I would like students to learn backtracking by writing their own versions of these algorithms.

A major component of the second semester will be algorithms for handling spoken language, including the Viterbi algorithm and the use of HMMs. I would include backtracking, the Viterbi algorithm and the use of HMMs in the category of general purpose algorithms that all of our students should learn before graduating. The actual spoken language understanding and generation will be done with packages. I would also like to include the use of VoiceXML as a easy way for students to build an interactive system.

Finally, I would also like to find a good application for introducing constraint satisfaction, as I feel that this is a valuable technique that is not often taught.

I know students would like to do some game development, perhaps using Pygame. While I would like them to learn the basics of event-driven programming, I am afraid that the details of game development would overwhelm the value of the assignment.

There are also additional NLP topics available in the NLTK book and elsewhere that I have not yet tried to teach in this format, including other types of grammars and statistical machine translation.

NIU scheduling for the M.S. program works best when students can start in any semester. For this reason the second course needs to be independent of the first rather than requiring it as a prerequisite. My proposed syllabus for the second course contains about three weeks of overlap at the beginning to make it available to students with no Python background. Since one of these weeks covers programming style and functional programming, which students always need more practice in, there are only two weeks of overlap. At that point it is reasonable for students to receive credit for both courses.

Conclusions

This paper describes a syllabus for teaching NLP to computer science majors with no background in linguistics. The course includes carefully sequenced Python programming assignments that teach Python programming, the fundamentals of NLP, and some algorithms every computer scientist should know in an organized fashion. This course has been successfully taught to undergraduates with a strong programming background as well as to international graduate students with a wide range of undergraduate preparation. They have enjoyed learning Python and have learned something new about language processing. I have enjoyed keeping up with natural language processing in a way that is only possible from detailed programming. Both of us have enjoyed having novel assignments in a programming class.

Acknowledgments

I thank the authors of NLTK and the NLTK book (Bird, Klein, and Loper 2009), whose extensive labor has made it possible to teach this course. I also thank the reviewers for their detailed and helpful comments.

References

Bird, S., Klein, E., and Loper, E. 2009. *Natural Language Processing in Python*. Sebastopol, CA: O'Reilly. Also available at http://www.nltk.org/book.

Freedman, R. 2005. Concrete Assignments for Teaching NLP in an M.S. Program. In Proceedings of the Second ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL, 37–42. Stroudsburg, PA: Association for Computational Linguistics.

Freedman, R. 2008. Teaching NLP to Computer Science Majors via Applications and Experiments. In Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics, 114–119. Stroudsburg, PA: Association for Computational Linguistics.

Jurafsky, D. and Martin, J. H. 2009. *Speech and Language Processing,* 2/e. Upper Saddle River, NJ: Prentice-Hall.

Kernighan, M., Church, K. W., and Gale, W. A. 1990. A spelling correction program based on a noisy channel model. In Proceedings of the 13th International Conference on Computational Linguistics (COLING), 2: 205–211. Stroudsburg, PA: Association for Computational Linguistics.

Pascoe, B. 2005. Webworks FSA applet. Available at http://www.cs.montana.edu/webworks/projects/theoryportal/models/fsa-exercise/appletCode/fsa_applet.html.

Porter, M. F. 1980. An algorithm for suffix stripping. *Program* 14(3): 130–137.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. San Francisco: Morgan Kaufmann.

Salton, G., and Buckley, C. 1988. Term-weighting approaches in automatic text retrieval. *Information Processing and Management* 24(5): 513-523.

Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory*. New York: Springer.

Weizenbaum, J. 1966. Eliza—A Computer Program for the Study of Natural Language Computation between Man and Machine. *Communications of the ACM* 9(1): 36–45.