Teaching Applied Natural Language Processing: Triumphs and Tribulations

Marti Hearst
School of Information Management & Systems
University of California, Berkeley
Berkeley, CA 94720
hearst@sims.berkeley.edu

Abstract

In Fall 2004 I introduced a new course called Applied Natural Language Processing, in which students acquire an understanding of which text analysis techniques are currently feasible for practical applications. The class was intended for interdisciplinary students with a somewhat technical background. This paper describes the topics covered and the programming exercises, emphasizing which aspects were successful and which problematic, and makes recommendations for future versions of the course.

1 Introduction

In Fall 2005 I introduced a new graduate level course called Applied Natural Language Processing.[1] The goal of this course was to acquaint students with the state-of-the-art of the field of NLP with an emphasis on applications. The intention was for students to leave the class with an understanding of what is currently feasible (and just on the horizon) to expect from content analysis, and how to use and extend existing NLP tools and technology. The course did not emphasize the theoretical underpinnings of NLP, although we did cover the most important algorithms. A companion graduate course on Statistical NLP was taught by Dan Klein in the Computer Science department. Dan's course focused on foundations and core NLP algorithms. Several computer science students took both courses, and thus learned both the theoretical and the applied sides of NLP. Dan and I discussed the goals and content of our respective courses in advance, but developed the courses independently.

2 Course Role within the SIMS Program

The primary target audience of the Applied NLP course were masters students, and to a lesser extent, PhD students, in the School of Information Management and Systems. (Nevertheless, PhD students in computer science and other fields also took the course.) MIMS students (as the SIMS masters students are known) pursue a professional degree studying information at the intersection of technology and social sciences. The students' technical backgrounds vary widely; each year a significant fraction have Computer Science undergraduate degrees, and another significant fraction have social science or humanities backgrounds. All students have an interest in technology and are required to take some challenging technical courses, but most non-CS background students are uncomfortable with advanced mathematics and are not as comfortable with coding as CS students are.

A key aspect of the program is the capstone final project, completed in the last semester, that (ideally) combines knowledge and skills obtained from throughout the program. Most students form a team of 3-4 students and build a system, usually to meet the requirements of an outside client or customer (although some students write policy papers and others get involved in research with faculty mem-

[1] Lecture notes, assignments, and other resources can be found at http://www.sims.berkeley.edu/courses/is290-2/f04/ .

Proceedings of the Second ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL, pages 1–8,
Ann Arbor, June 2005.

bers). Often the execution of these projects makes use of user-centered design, including a needs assessment, and iterative design and testing of the artifact. These projects often also have a backend design component using database design principles, document engineering modeling, or information architecture and organization principles, with sensitivity to legal considerations for privacy and intellectual property. Students are required to present their work to an audience of students, faculty, and professionals, produce a written report, and produce a website that describes and demonstrates their work.

In many cases these projects would benefit greatly from content analysis. Past projects have included a system to query on and monitor news topics as they occur across time and sources, a system to analyze when and where company names are mentioned in text and graph interconnections among them, a system to allow customization of news channels by topic, and systems to search and analyze blogs. Our past course offerings in this space focused on information retrieval with very little emphasis on content analysis, so students were using only IR-type techniques for these projects.

The state of the art in NLP had advanced sufficiently that the available tools can be employed for a number of projects like these. Furthermore, it is important for students attempting such projects to have an understanding of what is currently feasible and what is too ambitious. In fact, I find that this is a key aspect of teaching an applied class: learning what is possible with existing tools, what is feasible but requires more expertise than can be engineered in a semester with existing tools, and what is beyond the scope of current techniques.

3 Choosing Tools and Readings

The main challenges for a hands-on course as I'd envisioned surrounded finding usable interoperable tools, and defining feasible assignments that make use of programming without letting it interfere with learning.

There is of course the inevitable decision of which programming language(s) to work with. Scripting tools such as python are fast and easy to prototype with, but require the students to learn a new programming language. Java is attractive because many tools are written in it and the MIMS students were familiar with java – they are required to use it for two of their required courses but still tend to struggle with it. I did not consider perl since python is a more principled language and is growing in acceptance and in tool availability.

In the end I decided to require the students to learn python because I wanted to use NLTK, the Natural Language Toolkit (Loper and Bird, 2002). One goal of NLTK is to remove the emphasis on programming to enable students to achieve results quickly; and this aligned with my primary goal. NLTK seemed promising because it contained some well-written tutorials on n-grams, POS tagging and chunking, and contained text categorization modules. (I also wanted support for entity extraction, which NLTK does not supply.) NLTK is written in python, and so I decided to try it and have the students learn a new programming language. As will be described in detail below, our use of NLTK was somewhat successful, but we experienced numerous problems as well.

I made a rather large mistake early on by not spending time introducing python, since I wanted the assignments to correspond to the lectures and did not want to spend lecture time on the programming language itself. I instructed students who had registered for the course to learn python during the summer, but (not surprisingly) many of did not and had to struggle in the first few weeks. In retrospect, I realize I should have allowed time for people to learn python, perhaps via a lab session that met only during the first few weeks of class.

Another sticking point was student exposure to regular expressions. Regex's were very important and useful practical tools both for tokenization assignments and for shallow parsing. I assumed that the MIMS students had gotten practice with regular expressions because they are required to take a computer concepts foundations course which I designed several years ago. Unfortunately, the lecturer who took over the class from me had decided to omit regex's and related topics. I realized that I had to do some remedial coverage of the topic, which of course bored the CS students and which was not complete enough for the MIMS students. Again this suggests that perhaps some kind of lab is needed for getting people caught up in topics, or that perhaps

the first few weeks of the class should be optional for more advanced students.

I was also unable to find an appropriate textbook. Neither Schütze & Manning nor Jurafsky & Martin focus on the right topics. The closest in terms of topic is *Natural Language Processing for Online Applications* by Peter Jackson & Isabelle Moulinier, but much of this book focuses on Information Retrieval (which we teach in two other courses) and did not go into depth on the topics I most cared about. Instead of a text, students read a small selection of research papers and the NLTK tutorials.

4 Topics

The course met twice weekly for 80 minute periods. The topic coverage is shown below; topics followed by (2) indicate two lecture periods were needed.

Course Introduction
Using Large Collections (intro to NLTK)
Tokenization, Morphological Analysis
Part-of-Speech Tagging
Conditional Probabilities
Shallow Parsing (2)
Text Classification: Introduction
Text Classification: Feature Selection
Text Classification: Algorithms
Text Classification: Using Weka
Information Extraction (2)
Email and Anti-Spam Analysis
Text Data Mining
Lexicons and Ontologies
FrameNet (guest lecture by Chuck Fillmore)
Enron email dataset (in-class work) (2)
Spelling Correction / Clustering
Summarization (guest lecture by Drago Radev)
Question Answering (2)
Machine Translation (slides by Kevin Knight)
Topic Segmentation / Discourse Processing
Class Presentations

Note the lack of coverage of full syntactic parsing, which is covered extensively in Dan Klein's course. I touched on it briefly in the second shallow parsing lecture and felt this level of coverage was acceptable because shallow parsing is often as useful if not more so than full parsing for most applications. Note also the lack of coverage of word sense disambiguation. This topic is rich in algorithms, but was omitted primarily due to time constraints, but in part because of the lack of well-known applications.

Based on the kinds of capstone projects the MIMS students have done in the past, I knew that the most important techniques for their needs surrounded text categorization and information extraction/entity recognition. There are terrific software resources for text categorization and the field is fairly mature, so I had my PhD students Preslav Nakov and Barbara Rosario gave the lectures on this topic, in order to provide them with teaching experience.

The functionality provided by named entity recognition is very important for a wide range of real-world applications. Unfortunately, none of the free tools that we tried were particularly successful. Those that are available are difficult to configure and get running in a short amount of time, and have virtually no documentation. Furthermore, the state-of-the-art in algorithms is not present in the available tools in the way that more mature technologies such as POS tagging, parsing, and categorization are.

5 Using NLTK

5.1 Benefits

We used the latest version of NLTK, which at the time was version 1.4.[2] NLTK supplies some preprocessed text collections, which are quite useful. (Unfortunately, the different corpora have different types of preprocessing applied to them, which often lead to confusion and extra work for the class.) The NLTK tokenizer, POS taggers and the shallow parser (chunker) have terrific functionality once they are understood; some students were able to get quite accurate results using these and the supplied training sets. The ability to combine different n-gram taggers within the structure of a backoff tagger also supported an excellent exercise. However, a somewhat minor problem with the taggers is that there is no compact way to store the model resulting from tagging for later use. A serialized object could be created and stored, but the size of such object was so large that it takes about as long to load it into memory as it does to retrain the tagger.

[2] http://nltk.sourceforge.org

5.2 Drawbacks

There were four major problems with NLTK from the perspective of this course. The first major problem was the inconsistency in the different releases of code, both in terms of incompatibilities between the data structures in the different versions, and incompatibility of the documentation and tutorials within the different versions. It was tricky to determine which documentation was associated with which code version. And much of the contributed code did not work with the current version.

The second major problem was related to the first, but threw a major wrench into our plans: some of the advertised functionality simply was not available in the current version of the software. Notably, NLTK advertised a text categorization module; without this I would not have adopted NLTK as the coding platform for the class. Unfortunately, the most current version did not in fact support categorization, and we discovered this just days before we were to begin covering this topic.

The third major problem was the incompleteness of the documentation for much of the code. This to some degree undermined the goal of reducing the amount of work for students, since they (and I) had to struggle to figure out what was going on in the code and data structures.

One of these documentation problems centered around the data structure for conditional probabilities. NLTK creates a FreqDist class which is explained well in the documentation (it records a count for each occurrence of some phenomenon, much like a hash table) and provides methods for retrieving the max, the count and frequency of each occurrence, and so on. It also provides a class called a CondFreqDist, but does not document its methods nor explain its implementation. Users have to scrutinize the examples given and try to reverse engineer the data structure. Eventually I realized that it is simply a list of objects of type FreqDist, but this was difficult to determine at first, and caused much wasting of time and confusion among the students. There is also confusion surrounding the use of the method names *count* and *frequency* for FreqDist. Count refers to number of occurrences and frequency to a probability distribution across items, but this distinction is never stated explicitly although it can be inferred from a table of methods in the tutorial.

A less dramatic but still hampering problem was with the design of the core data structures, which make use of attribute tags rather than classes. This leads to rather awkward code structures. For example, after a sentence is tokenized, the results of tokenization are appended to the sentence data structure and are accessed via use of a subtoken keyword such as 'TOKENS'. To then run a POS tagger over the tokenized results, the 'TOKENS' keyword has to be specified as the value for a SUBTOKENS attribute, and another keyword must be supplied to act as the name of the tagged results. In my opinion it would be better to use the class system and define objects of different types and operations on those objects.

6 Assignments

One of the major goals of the class was for the students to obtain hands-on experience using and extending existing NLP tools. This was accomplished through a series of homework assignments and a final project. My pedagogical philosophy surrounding assignments is to supply as much as the functionality as necessary so that the coding that students do leads directly to learning. Thus, I try to avoid making students deal with details of formatting files and so on. I also try to give students a starting point to build up on.

The first assignment made use of some exercises from the NLTK tutorials. Students completed tokenizing exercises which required the use of the NLTK corpus tool accessors and the FreqDist and CondFreqDist classes. They also did POS tagging exercises which exposed them to the idea of n-grams, backoff algorithms, and to the process of training and testing. This assignment was challenging (especially because of some misleading text in the tagging tutorial, which has since been fixed) but the students learned a great deal. As mentioned above, I should have begun with a preliminary assignment which got students familiar with python basics before attempting this assignment.

For assignment 2, I provided a simple set of regular expression grammar rules for the shallow parser class, and asked the students to improve on these. After building the chunker, students were asked to

choose a verb and then analyze verb-argument structure (they were provided with two relevant papers (Church and Hanks, 1990; Chklovski and Pantel, 2004)). As mentioned above, most of the MIMS students were not familiar with regular expressions, so I should have done a longer unit on this topic, at the expense of boring the CS students.

The students learned a great deal from working to improve the grammar rules, but the verb-argument analysis portion was not particularly successful, in part because the corpus analyzed was too small to yield many sentences for a given verb and because we did not have code to automatically find regularities about the semantics of the arguments of the verbs. Other causes of difficulty were the students' lack of linguistic background, and the fact that the chunking part took longer than I expected, leaving students little time for the analysis portion of the assignment.

Assignments 3 and 4 are described in the following subsections.

6.1 Text Categorization Assignment

As mentioned above, text categorization is useful for a wide range SIMS applications, and we made it a centerpiece of the course. Unfortunately, we had to make a mid-course correction when I suddenly realized that text categorization was no longer available in NLTK.

After looking at a number of tools, we decided to use the Weka toolkit for categorization (Witten and Frank, 2000). We did not want the students to feel they had wasted their time learning python and NLTK, so we decided to make it easy for the students to reuse their python code by providing an interface between it and Weka.

My PhD student Preslav Nakov provided great help by writing code to translate the output of our python code into the input format expected by Weka. (Weka is written in java but has command line and GUI interfaces, and can read in input files and store models as output files.) As time went on we added increasingly more functionality to this code, tying it in with the NLTK modules so that the students could use the NLTK corpora for training and testing.[3]

Both Preslav and I had used Weka in the past but mainly with the command-line interface, and not taking advantage of its rich functionality. As with NLTK, the documentation for Weka was incomplete and out of date, and it was difficult to determine how to use the more advanced features. We performed extended experimentation with the system and developed a detailed tutorial on how to use the system; this tutorial should be of general use.[4]

For the categorization task, we used the "twenty newsgroups" collection that was supplied with NLTK. Unfortunately, it was not preprocessed into sentences, so I also had to write some sentence splitting code (based on Palmer and Hearst (1997)) so students could make use of their tokenizer and tagger code.

We selected one pair of newsgroups which contained very different content (*rec.motorcycles vs. sci.space*). We called this the diverse set. We then created two groups of newsgroups with more homogeneous content (a) *rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey*, and (b) *sci.crypt, sci.electronics, sci.med.original, sci.space*. The intention was to show the students that it is easier to automatically distinguish the heterogeneous groups than the homogeneous ones.

We set up the code to allow students to adjust the size of their training and development sets, and to separate out a reserved test set that would be used for comparing students' solutions.

We challenged the students to get the best scores possible on the held out test set, telling them not to use this test set until they were completely finished training and testing on the development set. (We relied on the honor system for this.) We made it known that we would announce which were the top-scoring assignments. As a general rule I avoid competition in my classes, but this was kept very low-key; only the top-scoring results would be named. Furthermore, innovative approaches that perhaps did not do as well as some others were also highlighted. Students were required to try at least 2 different types of features and 3 different classifiers.

This assignment was quite successful, as the stu-

[3] Available at http://www.sims.berkeley.edu/courses/is290-2/f04/assignments/assignment3.html

[4] Available at http://www.sims.berkeley.edu/courses/is290-2/f04/lectures/lecture11.ppt

dents were creative about building their features, and it was possible to achieve very strong results (much stronger than I expected) on both sets of newsgroups. The best scoring approaches got 99% accuracy on the 2-way diverse distinction and 97% accuracy on the 4-way homogeneous distinction.

6.2 Enron Email Assignment

Many of the SIMS students are interested in social networking and related topics. I decided as part of the class that we would analyze a relatively new text collection that had become available and that contained the potential for interesting text mining and analysis. I was also interested in having the class help produce a resource that would be of use to other classes and researchers. Thus we decided to take on the Enron email corpus,[5] on which limited analysis had been done.

My PhD student Andrew Fiore wrote code to preprocess this text, removing redundancies, normalizing email addresses, labeling quoted text, and so on. He and I designed a database schema for representing much of the structure of the collection and loaded in the parsed text. I created a Lucene[6] index for doing free text queries while Andrew built a highly functional web interface for searching fielded components. Andrew's system eventually allowed for individual students to login and register annotations on the email messages.

This collection consists of approximately 200,000 messages after the duplicates have been removed. We wanted to identify a subset of emails that might be interesting for analysis while at the same time avoiding highly personal messages, messages consisting mainly of jokes, and so on. After doing numerous searches, we decided to try to focus primarily on documents relating to the California energy crisis, trading discrepancies, and messages occurring near the end of the time range (just before the company's stock crashed).

After selecting about 1500 messages, I devised an initial set of categories. In class we refined these. One student had the interesting idea of trying to identify change in emotional tone as the scandals surrounding the company came to light, so we added emotional tone as a category type. Each message was then read and annotated by two students using the pre-defined categories. Students were asked to reconcile their differences when they had them.

Despite these safeguards, my impression is that the resulting assignments are far from consistent and the categories themselves are still rather ad hoc and oftentimes overlapping. There were many difficult curation issues, such as how to categorize a message with forwarded content when that content differed in kind from the new material. If we'd spent more time on this we could have done a better job, but as this was not an information organization course, I felt we could not spend more time on perfecting the labels. Thus, I do not recommend the category labels be used for serious analysis. Nevertheless, a number of researchers have asked for the cleaned up database and categories, and we have made them publicly available, along with the search interface.[7]

The students were then given two weeks to process the collection in some manner. I made several suggestions, including trying to automatically assign the hand-assigned categories, extending some automatic acronym recognition work that we'd done in our research (Schwartz and Hearst, 2003), using named entity recognition code to identify various actors, clustering the collection, or doing some kind of social network analysis. Students were told that they could extend this assignment into their final projects if they chose.

For most students it was difficult to obtain a strong result using this collection. The significant exception was for those students who worked on extending our acronym recognition algorithm; these projects were quite successful. (In fact, one student managed to improve on our results with a rather simple modification to our code.) Students often had creative ideas that were stymied by the poor quality of the available tools. Two groups used the MALLET named entity recognizer toolkit[8] in order to do various kinds of social network analysis, but the results were poor. (Students managed to make up for this deficiency in creative ways.)

I was a bit worried about students trying to use clustering to analyze the results, given the general difficulty of making sense of the results of cluster-

[5] http://www-2.cs.cmu.edu/ enron/

[6] http://lucene.apache.org

[7] http://bailando.sims.berkeley.edu/enron_email.html

[8] http://mallet.cs.umass.edu

ing, and this concern was justified. Clustering based on Weka and other tools is of course memory- and compute-intensive, but more problematically, the results are difficult to interpret. I would recommend against allowing students to do a text clustering exercise unless within a more constrained environment.

In summary, students were excited about building a resource based on relatively untapped and very interesting data. The resulting analysis on this untamed text was somewhat disappointing, but given that only two weeks were spent on this part of the assignment, I believe it was a good learning experience. Furthermore, the resulting resource seems to be of interest to a number of researchers, as was our intention.

6.3 Final Projects

I deliberately kept the time for the final projects short (about 3 weeks) so students would not go overboard or feel pressure to do something hugely time-consuming. The goal was to allow students to tie together some of the different ideas and skills they'd acquired in the class (and elsewhere), and to learn them in more depth by applying them to a topic of personal interest.

Students were encouraged to work in pairs, and I suggested a list of project ideas. Students who adopted suggested projects tended to be more successful than those who developed their own. Those who tried other topics were often too ambitious and had trouble getting meaningful results. However, several of those students were trying ideas that they planned to apply to their capstone projects, and so it was highly valuable for them to get a preview of what worked and what did not.

One suggestion I made was to create a back-of-the-book indexer, specifically for a recipe book, and one team did a good job with this project. Another was to improve on or apply an automatic hierarchy generation tool that we have developed in our research (Stoica and Hearst, 2004). Students working on a project to collect metadata for camera phone images successfully applied this tool to this problem. Again, social networking analysis topics were popular but not particularly successful; NLP tools are not advanced enough yet to meet the needs of this intriguing topic area. Not surprisingly, when students started with a new (interesting) text collection, they were bogged down in the preprocessing stage before they could get much interesting work done.

6.4 Reflecting on Assignments

Although students were excited about the Enron collection and we created a resource that is actively being used by other researchers, I think in future versions of the class I will omit this kind of assignment and have the students start their final projects sooner. This will allow them time to do any preprocessing necessary to get the text into shape for doing the interesting work. I will also exercise more control over what they are allowed to attempt (which is not my usual style) in order to ensure more successful outcomes.

I am not sure if I will use NLTK again or not. If the designers make significant improvements on the code and documentation, then I probably will. The style and intent of the tutorials are quite appropriate for the goals of the class. Students with stronger coding background tended to use java for their final projects, whereas the others tended to build on the python code we developed in the class assignments, which suggests that this kind of toolkit approach is useful for them.

7 Conclusions

Overall, I feel the main goals of the course were met. Although I am emphasizing how the course could be improved, most students were quite positive about the class, giving it an overall score of 5.8 out of 7 with a mode of 6 in their anonymous course reviews. (This is on the low side for my courses; most who gave it low scores found the programming too difficult.)

Most students found the material highly stimulating and the work challenging but not overwhelming. Several students mentioned that a lab session with a dedicated TA would have been desirable. Several suggested covering less material in more depth and several commented that the Enron exercise was a neat idea although not entirely successful in execution. Students remarked on liking reading research papers rather than a textbook (they also liked the relatively light reading load, which I feel was appropriate given the heavy assignment load). Some students

wanted more emphasis on real-world applications; I think it would be useful to have guest speakers from industry talk about this if possible.

I would like to see more research tools developed to a point to which they can be applied more successfully, especially in the area of information extraction. I would also recommend to colleagues that careful control be retained over assignments and projects to ensure feasibility in the outcome. It is more difficult to get good results on class projects in NLP than in other areas I've taught. As we so often see in text analysis work, it can often be difficult to do better than simple word counts for many projects.

I am interested in hearing ideas about how to accommodate both the somewhat technical and the highly technical students, especially in the early parts of the course. Perhaps the best solution is to offer an optional laboratory section, at least for the first few weeks, but perhaps for the entire term, but this solution obviously requires more resources.

When designing this course I did a fairly extensive web search looking for courses that offered what I was interested in, but didn't find much. I used the proceedings of the ACL-02 workshop on teaching NLP (where I learned about NLTK) as well as the NLP Universe. I think it would be a good idea to start an archive of teaching resources; ACM SIGCHI is in the midst of creating such an educational digital library and this example is worth studying.[9]

Acknowledgements

Thanks to Preslav Nakov, Andrew Fiore, and Barbara Rosario for their help with the class, and for all the students who took the class. Thanks also to Steven Bird and Edward Loper for developing and sharing NLTK, and for their generous time and help with the system during the course of the class. This work was supported in part by NSF DBI-0317510.

References

Timothy Chklovski and Patrick Pantel. 2004. Verbocean: Mining the web for fine-grained semantic verb relations. In *Proceedings of EMNLP*, Barcelona.

Kenneth W. Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *American Journal of Computational Linguistics*, 16(1):22–29.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, Philadelphia.

David Palmer and Marti A. Hearst. 1997. Adaptive multilingual sentence boundary disambiguation. *Computational Lingiustics*, 23(2).

Ariel Schwartz and Marti Hearst. 2003. A simple algorithm for identifying abbreviation definitions in biomedical text. In *Proceedings of the Pacific Symposium on Biocomputing (PSB 2003)*, Kauai, Hawaii.

Emilia Stoica and Marti Hearst. 2004. Nearly-automated metadata hierarchy creation. In *Proceedings of HLT-NAACL Companion Volume*, Boston.

Ian H. Witten and Eibe Frank. 2000. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco.

[9] http://hcc.cc.gatech.edu/