ACL-05

Second Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics

Proceedings

25 June 2005
University of Michigan
Ann Arbor, Michigan, USA

Production and Manufacturing by
Omnipress Inc.
Post Office Box 7214
Madison, WI 53707-7214

Introduction

This is the second workshop on effective tools and methodologies for teaching NLP and Computational Linguistics. Recurring themes are the interdisciplinarity of the subject and the unexpected mismatches between student background and what the instructor initially assumed. Some papers talk about particular tools and assignments: are spelling checkers really as cool as we think? how much can students be persuaded to like error analysis? is it fair to ask students to use research software when they could be learning about more generally applicable tools? Others describe teaching approaches conditioned by particular institutional and educational needs: what is it like teaching in a situation where students typically can't afford to buy even one textbook? can multiple universities co-ordinate a continent-wide curriculum? There is a consistent and welcome effort to discuss not only successes but also decisions that seem, at least in retrospect, to have been mistakes. We hope that all aspects of the sharing of experience begun in the papers will continue during the workshop itself.

We are grateful to all the authors who submitted papers, and the following people, who served on the program committee.

Steven Bird, University of Melbourne, Australia
Chris Brew, Ohio State University, USA
Ted Briscoe, University of Cambridge, UK
Walter Daelemans, University of Antwerp, Belgium
Robert Dale, Macquarie University, Australia
Jason Eisner, Johns Hopkins University, USA
Tomaž Erjavec, Jožef Stefan Institute, Slovenia
Kathy McKeown, Columbia University, USA
Elizabeth Liddy, Syracuse University, USA
Chris Manning, Stanford University, USA
Jim Martin, University of Colorado, USA
Chris Mellish, University of Aberdeen, UK
Dragomir Radev, University of Michigan, USA
Ellen Riloff, University of Utah, USA
Anoop Sarkar, Simon Fraser University, Canada
Harold Somers, University of Manchester, UK
Richard Sproat, University of Illinois, USA
Matthew Stone, Rutgers University, USA
Josef van Genabith, Dublin City University, Ireland
Richard Wicentowski, Swarthmore College, USA

We are also grateful to our home institutions for encouragement and support in working on our own teaching.

Chris Brew and Dragomir Radev

Table of Contents

Conference Program

Saturday, June 25, 2005

09:00–09:20	Opening session
	Session 1: NLP/CL in the curriculum
09:20-09:40	*Teaching Applied Natural Language Processing: Triumphs and Tribulations* Marti Hearst
09:40-10:00	*Teaching Dialogue to Interdisciplinary Teams through Toolkits* Justine Cassell and Matthew Stone
10:00-10:20	*"Language and Computers": Creating an Introduction for a General Undergraduate Audience* Chris Brew, Markus Dickinson and W. Detmar Meurers
10:30–11:00	Break
	Session 2: Assignments and tools
11:00–11:20	*A Core-Tools Statistical NLP Course* Dan Klein
11:20–11:40	*Web-based Interfaces for Natural Language Processing Tools* Marc Light, Robert Arens and Xin Lu
11:40–12:00	*Making Hidden Markov Models More Transparent* Nashira Lincoln and Marc Light
12:00–12:20	*Concrete Assignments for Teaching NLP in an M.S. Program* Reva Freedman
12:30–14:00	Lunch

Saturday, June 25, 2005 (continued)

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.

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

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 members). 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]

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

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-

[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/

Teaching Dialogue to Interdisciplinary Teams through Toolkits

Justine Cassell
Technology and Social Behavior
Northwestern University
justine@northwestern.edu

Matthew Stone
Computer Science and Cognitive Science
Rutgers University
matthew.stone@rutgers.edu

Abstract

We present some lessons we have learned from using software infrastructure to support coursework in natural language dialogue and embodied conversational agents. We have a new appreciation for the differences between coursework and research infrastructure—supporting teaching may be harder, because students require a broader spectrum of implementation, a faster learning curve and the ability to explore mistaken ideas as well as promising ones. We outline the collaborative discussion and effort we think is required to create better teaching infrastructure in the future.

1 Introduction

Hands-on interaction with dialogue systems is a necessary component of a course on computational linguistics and natural language technology. And yet, it is clearly impracticable to have students in a quarter-long or semester-long course build a dialogue system from scratch. For this reason, instructors of these courses have experimented with various options to allow students to view the code of a working dialogue system, tweak code, or build their own application using a dialogue system toolkit. Some popular options include the NLTK (Loper and Bird, 2002), CSLU (Cole, 1999), Trindi (Larsson and Traum, 2000) and Regulus (Rayner et al., 2003) toolkits. However, each of these options has turned out to have disadvantages. Some of the toolkits require too much knowledge of linguistics for the average computer science student, and vice-versa, others require too much programming for the average linguist. What is needed is an extensible dialogue toolkit that allows easy application building for beginning students, and more sophisticated access to, and tweakability of, the models of discourse for advanced students.

In addition, as computational linguists become increasingly interested in the role of non-verbal behavior in discourse and dialogue, more of us would like to give our students exposure to models of the interaction between language and nonverbal behaviors such as eye gaze, head nods and hand gestures. However, the available dialogue system toolkits either have no graphical body or if they do have (part of) a body—as in the case of the CSLU toolkit—the toolkit does not allow the implementation of alternative models of body–language interaction.

We feel, therefore, that there is a need for a toolkit that allows the beginning graduate student—who may have some computer science or some linguistics background, but not both—to implement a working embodied dialogue system, as a way to experiment with models of discourse, dialogue, collaborative conversation and the interaction between verbal and nonverbal behavior in conversation. We believe the community as a whole must be engaged in the design, implementation and fielding of this kind of educational software. In this paper, we survey the experience that has led us to these conclusions and frame the broader discussion we hope the TNLP workshop will help to further.

2 Our Courses

Our perspective in this paper draws on more than fifteen course offerings at the graduate level in discourse and dialogue over the years. Justine Cassell's course *Theories and Technologies of Human Communication* is documented on the web here:

http://www.soc.northwestern.edu/justine/discourse

Matthew Stone's courses *Natural Language Processing* and *Meaning Machines*[1] are documented here:

http://www.cs.rutgers.edu/~mdstone/class/533-spring-03/
http://www.cs.rutgers.edu/~mdstone/class/672

These courses are similar in perspective. All address an extremely diverse and interdisciplinary audience of students from computer science, linguistics, cognitive science, information science, communication, and education. The typical student is a first or second-year PhD student with a serious interest in doing a dissertation on human-computer communication or in enriching their dissertation research with results from the theory or practice of discourse and dialogue. All are project courses, but no programming is required; projects may involve evaluation of existing implementations or the prospective design of new implementations based on ongoing empirical research. Nevertheless, the courses retain the dual goals that students should not only understand discourse and the theory of pragmatics, but should also understand how the theory is implemented, either well enough to talk intelligently about the implementation or, if they are computer scientists, to actually carry it out.

As befits our dual goals, our courses all involve a mix of instruction in human-human dialogue and human-computer dialogue. For example, Cassell begins her course with a homework where students collect, transcribe and analyze their own recordings of face-to-face conversation. Students are asked to discuss what constitutes a sufficient record of discourse, and to speculate on what the most challenging processing issues would be to allow a computer to replace one of the participants. Computer scientists definitely have difficulty with this aspect of the course—only fair, since they are at the advantage when it comes to implementation. But computer scientists see the value in the exercise: even if they do not believe that interfaces should be designed to act like people, they still recognize that well-designed interactive systems must be ready to handle the kinds of behaviors people actually carry out. And hands-on experience convinces them that behavior in human conversation is both rich and surprising. The computer scientists agree—after turning in impoverished and uninformed "analyses" of their discourse for a brutal critique—that they will never look at conversation the same way again.

Our experience suggests that we should be trying to give students outside computer science the same kind of eye-opening hands-on experience with technology. For example, we have found that linguists are just as challenged and excited by the discipline of technology as computer scientists are by the discipline of empirical observations. Linguists in our classes typically report that successful engagement with technology "exposes a lot of details that were missing from my theoretical understanding that I never would have considered without working through the code". Nothing is better at bringing out the assumptions you bring to an analysis of human-human conversation than the thought experiment of replacing one of the participants by something that has to struggle consciously to understand it—a space alien, perhaps, or, more realistically, an AI system. We are frustrated that no succinct assignment, comparable to our transcription homework, yet exists that can reliably deliver this insight to students outside computer science.

3 Framing the Problem

Our courses are not typical NLP classes. Our treatment of parsing is marginal, and for the most part we ignore the mainstays of statistical language processing courses: the low-level technology such as finite-state methods; the specific language processing challenges for machine learning methods; and "applied" subproblems like named entity extraction, or phrase chunking. Our focus is almost exclusively on high-level and interactional issues, such as the structure of discourse and dialogue, information structure, intentions, turn-taking, collaboration,

[1]The catchy title is the inspiration of Deb Roy at MIT.

reference and clarification. Context is central, and under that umbrella we explicitly discuss both the perceptual environment in which conversation takes place and the non-verbal actions that contribute to the management of conversation and participants' real-world collaborations.

Our unusual focus means that we can not readily take advantage of software toolkits such as NLTK (Loper and Bird, 2002) or Regulus (Rayner et al., 2003). These toolkits are great at helping students implement and visualize the fundamentals of natural language processing—lexicon, morphology, syntax. They make it easy to experiment with machine learning or with specific models for a small scale, short course assignment in a specific NLP module. You can think of this as a “horizontal” approach, allowing students to systematically develop a comprehensive approach to a single processing task. But what we need is a “vertical” approach, which allows students to follow a specific choice about the representation of communicative behaviors or communicative functions all the way through an end-to-end dialogue system. We have not succeeded in conceptualizing how a carefully modularized toolkit would support this kind of student experience.

Still, we have not met with success with alternative approaches, either. As we describe in Section 3.1, our own research systems may allow the kinds of experiments we want students to carry out. But they demand too much expertise of students for a one-semester course. In fact, as we describe in Section 3.2, even broad research systems that come with specific support for students to carry out a range of tasks may not enable the specific directions that really turn students on to the challenge of discourse and dialogue. However, our experience with implementing dedicated modules for teaching, as described in Section 3.3, is that the lack of synergy with ongoing research can result in impoverished tools that fail to engage students. We don't have the tools we want—but our experience argues that we think the tools we really want will be developed only through a collaborative effort shared across multiple sites and broadly engaged with a range of research issues as well as with pedagogical challenges.

3.1 Difficulties with REA and BEAT

Cassell has experimented with the use of her research platforms REA (Cassell et al., 1999) and BEAT (Cassell et al., 2001) for course projects in discourse and dialogue. REA is an embodied conversational agent that interacts with a user in a real estate agent domain. It includes an end-to-end dialogue architecture; it supports speech input, stereo vision input, conversational process including presence and turn-taking, content planning, the context-sensitive generation of communicative action and the animated realization of multimodal communicative actions. BEAT (the behavior expression animation toolkit), on the other hand, is a module that fits into animation systems. It marks up text to describe appropriate synchronized nonverbal behaviors and speech to realize on a humanoid talking character.

In teaching dialogue at MIT, Cassell invited students to adapt her existing REA and BEAT system to explore aspects of the theory and practice of discourse and dialogue. This led to a range of interesting projects. For example, students were able to explore hypothetical differences among characters—from virtual “Italians” with profuse gesture, to virtual children whose marked use of a large gesture space contrasted with typical adults, to characters who showed new and interesting behavior such as the repeated foot-tap of frustrated condescension. However, we think we can serve students much better. Many of these projects were accomplished only with substantial help from the instructor and TAs, who were already extremely familiar with the overall system. Students did not have time to learn how to make these changes entirely on their own.

The foot-tapping agent is a good example of this. To add foot-tapping is a paradigmatic “vertical” modification. It requires adding suitable context to the discourse state to represent uncooperative user behavior; it requires extending the process for generating communicative actions to detect this new state and schedule an appropriate behavioral response; and then it requires extending the animation platform to be able to show this behavior. BEAT makes the second step easy—as it should be—even for linguistics students. To handle the first and third steps, you would hope that an interdisciplinary team containing a communication student and a computer science student would be able to bring the expertise to design the new dialogue state and the new animated behavior. But that wasn't exactly true. In order to add the behavior to REA, students needed not only background in the relevant technology—like what a computer scientist would learn in a general human animation class. To add the behavior, students also needed to know how this technology was realized in our particular research platform. This proved too much for one semester.

We think this is a general problem with new research systems. For example, we think many of the same issues would arise in asking students to build a dialogue system on top of the Trindi toolkit in a one semester course.

3.2 Difficulties with the CSLU toolkit

In Fall 2004, Cassell experimented with using the CSLU dialogue toolkit (Cole, 1999) as a resource for class projects. This is a broad toolkit to support research and teaching in spoken language technology. A particular strength of the toolkit is its support for the design of finite-state dialogue models. Even students outside computer science appreciated the toolkit's drag-and-drop interface for scripting dialogue flow. For example, with this interface, you can add a repair sequence to a dialogue flow in one easy step. However, the indirection the toolkit places between students and the actual constructs of dialogue theory can by quite challenging. For example, the finite-state architecture of the CSLU toolkit allows students to look at floor management and at dialogue initiative only indirectly: specific transition networks encode specific strategies for taking turns or managing problem solving by scheduling specific communicative functions and behaviors.

The way we see it, the CSLU toolkit is more heavily geared towards the rapid construction of particular kinds of research prototypes than we would like in a teaching toolkit. Its dialogue models provide an instructive perspective on actions in discourse, one that nicely complements the perspective of DAMSL (Core and Allen, 1997) in seeing utterances as the combined realization of a specific, constrained range of communicative functions. But we would like to be able to explore a range of other metaphors for organizing the information in dialogue. We would like students to be able to realize models of face-to-face dialogue (Cassell et al., 2000), the information-state approach to domain-independent practical dialogue (Larsson and Traum, 2000), or approaches that emphasize the grounding of conversation in the specifics of a particular ongoing collaboration (Rich et al., 2001). The integration of a talking head into the CSLU toolkit epitomizes these limitations with the platform. The toolkit allows for the automatic realization of text with an animated spoken delivery, but does not expose the model to programmers, making it impossible for programmers adapt or control the behavior of the face and head.

We think this is a general problem with platforms that are primarily designed to streamline a particular research methodology. For example, we think many of the same issues would arise in asking students to build a multimodal behavior realization system on top of a general-purpose speech synthesis platform like Festival (Black and Taylor, 1997).

3.3 Difficulties with TAGLET

At this point, the right solution might seem to be to devise resources explicitly for teaching. In fact, Stone advocated more or less this at the 2002 TNLP workshop (2002). There, Stone motivated the potential role for a simple lexicalized formalism for natural language syntax, semantics and pragmatics in a broad NLP class whose emphasis is to introduce topics of current research.

The system, TAGLET, is a context-free tree-rewriting formalism, defined by the usual complementation operation and the simplest imaginable modification operation. This formalism may in fact be a good way to present computational linguistics to technically-minded cognitive science students—those rare students who come with interest and experience in the science of language as well as a solid ability to program. By implementing a strong competence TAGLET parser and generator students simultaneously get experience with central computer science ideas—data structures, unification, recursion and abstraction—and develop an effective starting point for their own subsequent projects.

However, in retrospect, TAGLET does not serve to introduce students outside computer science to the distinctive insights that come from a computational approach to language use. For one thing, to reach a broad audience, it is a mistake to focus on representations that programmers can easily build at the expense of representations that other students can easily understand. These other students need visualization; they need to be able to see what the system computes and how it computes it. Moreover, these other students can tolerate substantial complexity in the underlying algorithms if the system can be understood clearly and mechanistically in abstract terms. You wouldn't ask a computer scientist to implement a parser for full tree-adjoining grammar but that doesn't change the fact that it's still a perfectly natural, and comprehensible, algorithmic abstraction for characterizing linguistic structure.

Another set of representations and algorithms might avoid some of these problems. But a new approach could not avoid another problem that we think applies generally to platforms that are designed exclusively for teaching: there is no synergy with ongoing research efforts. Rich resources are so crucial to any computational treatment of dialogue: annotated corpora, wide-coverage grammars, plan-recognizers, context models, and the rest. We can't afford to start from scratch. We have found this concretely in our work. What got linguists involved in the computational exploration of dialogue semantics at Rutgers was not the special teaching resources Stone created. It was hooking students up with the systems that were being actively developed in ongoing research (DeVault et al., 2005). These research efforts made it practical to provide students with the visualizations, task and context models, and interactive architecture they needed to explore substantive issues in dialogue semantics. Whatever we do will have to closely connect teaching and our ongoing research.

4 Looking ahead

Our experience teaching dialogue to interdisciplinary teams through toolkits has been humbling. We have a new appreciation for the differences between coursework and research infrastructure—supporting teaching may be harder, because students require a broader spectrum of implementation, a faster learning curve and the ability to explore mistaken ideas as well as promising ones. But we increasingly think the community can and should come together to foster more broadly useful resources for teaching.

We have reframed our ongoing activities so that we can find new synergies between research and teaching. For example, we are currently working to expand the repertoire of animated action in our freely-available talking head RUTH (DeCarlo et al., 2004). In our next release, we expect to make different kinds of resources available than in the initial release. Originally, we distributed only the model we created. The next version will again provide that model, along with a broader and more useful inventory of facial expressions for it, but we also want the new RUTH to be more easily extensible than the last one. To do that, we have ported our model to a general-purpose animation environment (Alias Research's Maya) and created software tools that can output edited models into the collection of files that RUTH needs to run. This helps achieve our objective of quickly-learned extensibility. We expect that students with a background in human animation will bring experience with Maya to a dialogue course. (Anyway, learning Maya is much more general than learning RUTH!) Computer science students will thus find it easier to assist a team of communication and linguistics students in adding new expressions to an animated character.

Creating such resources to span a general system for face-to-face dialogue would be an enormous undertaking. It could happen only with broad input from those who teach discourse and dialogue, as we do, through a mix of theory and practice. We hope the TNLP workshop will spark this kind of process. We close with the questions we'd like to consider further. What kinds of classes on dialogue and discourse pragmatics are currently being offered? What kinds of audiences do others reach, what goals do they bring, and what do they teach them? What are the scientific and technological principles that others would use toolkits to teach and illustrate? In short, what would your dialogue toolkit make possible? And how can we work together to realize both our visions?

5 Acknowledgments

Thanks to Doug DeCarlo, NSF HLC 0308121.

References

Alan Black and Paul Taylor. 1997. Festival speech synthesis system. Technical Report HCRC/TR-83, Human Communication Research Center. http://www.cstr.ed.ac.uk/projects/festival/.

J. Cassell, T. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. Vilhjálmsson, and H. Yan. 1999. Embodiment in conversational characters: Rea. In *CHI 99*, pages 520–527.

Justine Cassell, Tim Bickmore, Lee Campbell, Hannes Vilhjalmsson, and Hao Yan. 2000. Human conversation as a system framework. In J. Cassell, J. Sullivan, S. Prevost, and E. Churchill, editors, *Embodied Conversational Agents*, pages 29–63. MIT Press, Cambridge, MA.

Justine Cassell, Hannes Vilhjálmsson, and Tim Bickmore. 2001. BEAT: the behavioral expression animation toolkit. In *SIGGRAPH*, pages 477–486.

Ron Cole. 1999. Tools for research and education in speech science. In *Proceedings of the International Conference of Phonetic Sciences*. http://cslu.cse.ogi.edu/toolkit/.

Mark G. Core and James F. Allen. 1997. Coding dialogs with the DAMSL annotation scheme. In *Working Notes of AAAI Fall Symposium on Communicative Action in Humans and Machines*. http://www.cs.rochester.edu/research/cisd/resources/damsl/.

Douglas DeCarlo, Corey Revilla, Matthew Stone, and Jennifer Venditti. 2004. Specifying and animating facial signals for discourse in embodied conversational agents. *Journal of Visualization and Computer Animation*. http://www.cs.rutgers.edu/˜village/ruth/.

David DeVault, Anubha Kothari, Natalia Kariaeva, Iris Oved, and Matthew Stone. 2005. An information-state approach to collaborative reference. In *ACL Proceedings Companion Volume (interactive poster and demonstration track)*. http://www.cs.rutgers.edu/˜mdstone/pointers/collabref.html.

Staffan Larsson and David Traum. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, 6:323–340. http://www.ling.gu.se/projekt/trindi/.

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*. http://nltk.sourceforge.net.

Manny Rayner, Beth Ann Hockey, and John Dowding. 2003. An open source environment for compiling typed unification grammars into speech recognisers. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computation Linguistics (interactive poster and demo track)*. http://sourceforge.net/projects/regulus.

C. Rich, C. L. Sidner, and N. Lesh. 2001. COLLAGEN: applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22:15–25.

Matthew Stone. 2002. Lexicalized grammar 101. In *ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, pages 76–83. http://www.cs.rutgers.edu/˜mdstone/class/taglet/.

"Language and Computers" Creating an Introduction for a General Undergraduate Audience

Chris Brew
Department of Linguistics
The Ohio State University
cbrew@ling.osu.edu

Markus Dickinson
Department of Linguistics
The Ohio State University
dickinso@ling.osu.edu

W. Detmar Meurers
Department of Linguistics
The Ohio State University
dm@ling.osu.edu

Abstract

This paper describes the creation of Language and Computers, a new course at the Ohio State University designed to be a broad overview of topics in computational linguistics, focusing on applications which have the most immediate relevance to students. This course satisfies the mathematical and logical analysis requirement at Ohio State by using natural language systems to motivate students to exercise and develop a range of basic skills in formal and computational analysis. In this paper we discuss the design of the course, focusing on the success we have had in offering it, as well as some of the difficulties we have faced.

1 Introduction

In the autumn of 2003, we created Language and Computers (Linguistics 384), a new course at the Ohio State University that is designed to be a broad overview of topics in computational linguistics, focusing on applications which have the most immediate relevance to students. Language and Computers is a general enrollment course designed to meet the Mathematical and Logical Analysis requirement that is mandated for all undergraduates at the Ohio State University (OSU), one of the largest universities in the US. We are committed to serving the average undergraduate student at OSU, including those for whom this is the first and last Linguistics course. Some of the students take the course because it is an alternative to calculus, others because of curiosity about the subject matter. The course was first taught in Winter 2004, drawing a wide range of majors, and has since expanded to three sections of up to 35 students each. In this paper we will discuss the design of the course, focusing on the success we have had in offering it, as well as some of the difficulties we have faced.

2 General Context

The Linguistics Department at OSU is the home of a leading graduate program in which 17 graduate students are currently specializing in computational linguistics. From the perspective of the graduate program, the goal of the new course development was to create more appropriate teaching opportunities for the graduate students specializing in computational linguistics. Much of the undergraduate teaching load in Linguistics at OSU is borne by graduate teaching assistants (GTAs) who receive stipends directly from the department. After a training course in the first year, most such GTAs act as instructors on the Department's "Introduction to Language," which is taught in multiple small sections. Instructors are given considerable responsibility for all aspects of course design, preparation, delivery, and grading. This works very well and produces many superb instructors, but by 2003 it was apparent that increasing competition was reducing the pool of undergraduates who want to take this general overview course.

The Ohio State University has a distribution requirement, the General Education Curriculum (GEC), that is designed to ensure adequate breadth in undergraduate education. The twin demands of the student's major and the distribution requirement are sufficient to take up the vast majority of the credit hours required for graduation. In practice this means that students tend to make course selections motivated primarily by the goal of completing the necessary requirements as quickly and efficiently as they can, possibly at the expense of curiosity-driven exploration. Linguistics, as an interdisciplnary subject, can create courses that satisfy both curiosity and GEC requirements.

To fill this interdisciplinary niche, the OSU Department of Linguistics has created a range of new courses such as Language and Gender, Language and the Mind, Language and the Law, and the Language and Computers course discussed in this paper. In addition to filling a distribution requirement niche for undergraduates, the courses also allow the linguistics GTAs to teach courses on topics that are related to their area of specialization, which can be beneficial both to the instructors and to those instructed. Prior to creation of the new Language and Computers course, there were virtually no opportunities for student members of the computational linguistics group to teach material close to their focus.

3 Course overview

The mission statement for our course reads:

> In the past decade, the widening use of computers has had a profound influence on the way ordinary people communicate, search and store information. For the overwhelming majority of people and situations, the natural vehicle for such information is natural language. Text and to a lesser extent speech are crucial encoding formats for the information revolution. This course will give students insight into the fundamentals of how computers are used to represent, process and organize textual and spoken information, as well as providing tips on how to effectively integrate this knowledge into their working practice. The course will cover the theory and practice of human language technology.

The course was designed to meet the Mathematical and Logical Analysis (MLA) requirement for students at the Ohio State University, which is characterized in the following way:

> A student in a B.A. program must take one course that focuses on argument in a context that emphasizes natural language, mathematics, computer science or quantitative applications not primarily involving data. Courses which emphasize the nature of correct argumentation either in natural languages or in symbolic form would satisfy this requirement, as would many mathematics or computer science courses. . . . The courses themselves should emphasize the logical processes involved in mathematics, inductive or deductive reasoning, or computing and the theory of algorithms.

Linguistics 384 responds to this specification by using natural language systems to motivate students to exercise and develop a range of basic skills in formal and computational analysis. The course combines lectures with group work and in-class discussions, resulting in a seminar-like environment. We enrol no more than 35 students per section, often significantly fewer at unpopular times of day.

The course philosophy is to ground abstract concepts in real world examples. We introduce strings, regular expressions, finite-state and context-free grammars, as well as algorithms defined over these structures and techniques for probing and evaluating systems that rely on these algorithms. This meets the MLA objective to emphasize the nature of correct argumentation in symbolic form as well as the logical processes involved in computing and the theory of algorithms. These abstract ideas are embedded in practical applications: web searching, spelling correction, machine translation and dialogue systems. By covering the technologies behind these applications, the course addresses the requirement to sharpen a student's ability to reason critically, construct valid arguments, think creatively, analyze objectively, assess evidence, perceive tacit assumptions, and weigh evidence.

Students have impressions about the quality of such systems, but the course goes beyond merely subjective evaluation of systems and emphasizes the use of formal reasoning to draw and argue for valid conclusions about the design, capabilities and behavior of natural language systems.

In ten weeks, we cover eight topics, using a data projector in class, with copies of the slides being handed out to the student before each class. There is no textbook, and there are relatively few assigned readings, as we have been unable to locate materials appropriate for an average student without required background who may never take another (computational) linguistics class. The topics covered are the following, in this order:

- Text and speech encoding
- (Web-)Searching
- Spam filtering (and other classification tasks, such as language identification)
- Writers' aids (Spelling and grammar correction)
- Machine translation (2 weeks)
- Dialogue systems (2 weeks)
- Computer-aided language learning
- Social context of language technology use

In contrast to the courses of which we are aware that offer computational linguistics to undergraduates, our Language and Computers is supposed to be accessible without prerequisites to students from every major (a requirement for GEC courses). For example, we cannot assume any linguistic background or language awareness. Like Lillian Lee's Cornell course (Lee, 2002), the course cannot presume programming ability. But the GEC regulations additionally prohibit us from requiring anything beyond high school level abilities in algebraic manipulation. We initially hoped that this meant that we would be able to rely on the kind of math knowledge that we ourselves acquired in secondary school, but soon found that this was not realistic. The sample questions from Lee's course seem to us to be designed for students who actively enjoy math. Our goal is different: we want to exercise and extend the math skills of the general student population, ensuring that the course is as accessible to the well-motivated dance major as it is to the geekier people with whom we are somewhat more familiar. This is hard, but worthwhile.

The primary emphasis is on discrete mathematics, especially with regard to strings and grammars. In addition, the text classification and spam-filtering component exercise the ability to reason clearly using probabilities. All of this can be achieved for students with no collegiate background in mathematics.

Specifically, Linguistics 384 uses non-trivial mathematics at a level at or just beyond algebra 1 in the following contexts:

- Reasoning about finite-state automata and regular expressions (in the contexts of web searching and of information management). Students reason about relationships between specific and general search terms.

- Reasoning about more elaborate syntactic representations (such as context-free grammars) and semantic representations (such as predicate calculus), in order to better understand grammar checking and machine translation errors.

- Reasoning about the interaction between components of natural language systems (in the contexts of machine translation and of dialog systems).

- Understanding the basics of dynamic programming via spelling correction (edit distance) and applying algebraic thinking to algorithm design.

- Simple probabilistic reasoning (in the context of text classification).

There is also an Honors version of the course, which is draws on a somewhat different pool of students. In 2004 the participants in Honors 384 were equally split between Linguistics majors looking for a challenging course, people with a computer background and some interest in language and people for whom the course was a good way of meeting the math requirement at Honors level. Most were seniors, so there was little feed-through to further Linguistics courses.

The Honors course, which used to be called Language Processing Technology, predates Language and Computers, and includes more hands-on material. Originally the first half of this course was an introduction to phonetics and speech acoustics through Praat, while the second was a Prolog-based introduction to symbolic NLP. We took the opportunity to redesign this course when we created the non-honors version. In the current regime, the hands-on aspect is less important than the opportunities offered by the extra motivation and ability of these students. Two reading assignments in the honors version were Malcolm Gladwell's book review on the Social Life of Paper (Gladwell, 2001) and Turing's famous paper on the Imitation Game (Turing, 1950). We wondered whether the eccentricity and dated language of the latter would be a problem, but it was not.

Practical assignments in the laboratory are possible in the honors course, because the class size can be limited. One such assignment was a straightforward run-through of the clock tutorial from the Festival speech synthesis system and another a little machine translation system between digits and number expressions. Having established that they can make a system that turns 24 into 'twenty four', and so on, the students are challenged to adapt it to speak "Fairy Tale English": that is, to make it translate 24 into 'four and twenty', and vice-versa.

1

4 General themes of the course

Across the eight different topics that are taught, we try to maintain a cohesive feel by emphasizing and repeating different themes in computational linguistics. Each theme allows the students to see that certain abstract ideas are quite powerful and can inform different concrete tasks. The themes which have been emphasized to this point are as follows:

- There are both statistical and rule-based methods for approaching a problem in natural language processing. We show this most clearly in the spam filtering unit and the machine translation unit with different types of systems.
- There is a tension between developing technology in linguistically-informed ways and developing technology so that a product is effective. In the context of dialogue systems, for example, the lack of any linguistic knowledge in ELIZA makes it fail quickly, but an ELIZA with a larger database and still no true linguistic knowledge could have more success.
- Certain general techniques, such as n-gram analysis, can be applied to different computational linguistic applications.
- Effective technology does not have to solve every problem; focusing on a limited domain is typically more practical for the applications we look at. In machine translation, this means that a machine translation system translating the weather (e.g., the METEO system) will perform better than a general-purpose system.
- Intelligent things are being done to improve natural language technology, but the task is a very difficult one, due to the complexities of language. Part of each unit is devoted to

[1]For a complete overview of the course materials, there are several course webpages to check out. The webpage for the first section of the course (Winter 2004) is at `http://ling.osu.edu/~dickinso/384/wi04/`. A more recent section (Winter 2005) can be found at `http://ling.osu.edu/~dm/05/winter/384/`. For the honors course, the most recent version is located at `http://ling.osu.edu/~cbrew/2005/spring/H384/`. A list of weblinks to demos, software, and on-line tutorials currently used in connection with the course can be found at `http://ling.osu.edu/~xflu/384/384links.html`

showing that the problem the technology is addressing is a complex one.

5 Aspects of the course that work

The course has been a positive experience, and students overall seemed pleased with it. This is based on the official student evaluation of instruction, anonymous, class specific questionnaires we handed out at the end of the class, personal feedback, and new students enrolling based on recommendations from students who took the course. We attribute the positive response to several different aspects of the course.

5.1 Topics they could relate to

Students seem to most enjoy those topics which were most relevant to their everyday life. On the technological end, this means that the units on spam filtering, web searching, and spell checking are generally the most well-received. The more practical the focus, the more they seem to appreciate it; for web searching, for instance, they tend to express interest in becoming better users of the web. On the linguistic end, discussions of how dialogue works and how language learning takes place, as part of the units on dialogue systems and CALL, respectively, tend to resonate with many students. These topics are only sketched out insofar as they were relevant to the NLP technology in question, but this has the advantage of not being too repetitive for the few students who have had an introductory linguistics class before.

5.2 Math they can understand

Students also seem to take pride in being able to solve what originally appear to be difficult mathematical concepts. To many, the concept and look of a binary number is alien, but they consistently find this to be fairly simple. The basics of finite-state automata and boolean expressions (even quite complicated expressions) provide opportunities for students to understand that they are capable of learning concepts of logical thinking. Students with more interest and more of an enjoyment for math are encouraged to go beyond the material and, e.g., figure out the nature of more complicated finite-state automata. In this way, more advanced students are able to stay interested without losing the other students.

More difficult topics, such as calculating the minimum edit distance between a word and its misspelling via dynamic programming, can be frustrating, but they just as often are a source of a greater feeling of success for students. After some in-class exercises, when it becomes apparent that the material is learnable and that there is a clear, well-motivated point to it, students generally seem pleased in conquering somewhat more difficult mathematical concepts.

5.3 Interactive demos

In-class demos of particular software are also usually well-received, in particular when they present applications that students themselves can use. These demos often focus on the end result of a product, such as simply listening to the output of several text-to-speech synthesizers, but they can also be used for understanding how the applications works. For example, some sections attempt to figure out as a class where a spelling checker fails and why. Likewise, an in-class discussion with ELIZA has been fairly popular, and students are able to deduce many of the internal properties of ELIZA.

5.4 Fun materials

In many ways, we have tried to keep the tone of the course fairly light. Even though we are teaching mathematical and logical concepts, these concepts are still connected to the real world, and as such, there is much opportunity to present the material in a fun and engaging manner.

Group work One such way to make the learning process more enjoyable was to use group work. In the past few quarters, we have been refining these exercises. Because of the nature of the topics, some topics are easier to derive group exercises for than others. The more mathematical topics, such as regular expressions, suit themselves well for straightforward group work on problem sets in class; others can be more creative. The group exercises usually serve as a way for students to think about issues they already know something about, often as a way to introduce the topic.

For example, on the first day, they are given a sheet and asked to evaluate sets of opposing claims, giving arguments for both sides, such as the following:

1. A person will have better-quality papers if they use a spell checker.

   A person will have worse-quality papers if they use a spell checker.

2. An English-German dictionary is the main component needed to automatically translate from English to German.

   An English-German dictionary is not the main component needed to automatically translate from English to German.

3. Computers can make you sound like a native speaker of another language.

   Computers cannot make you sound like a native speaker of another language.

To take another example, to get students thinking about the social aspects of the use of language technology, they are asked in groups to consider some of the implications of a particular technology. The following is an excerpt from one such handout.

> You work for a large software company and are in charge of a team of computational linguists. One day, you are told: "We'd like you and your team to develop a spell checker for us. Do you have any questions?" What questions do you have for your boss?
>
> ...
>
> Somehow or another, the details of your spell checker have been leaked to the public. This wouldn't be too bad, except that it's really ticked some linguists off. "It's just a big dictionary!" they yell. "It's like you didn't know anything about morphology or syntax or any of that good stuff." There's a rumor that they might sue you for defamation of linguistics. What do you do?

Although the premise is somewhat ridiculous, with such group work, students are able to consider important topics in a relaxed setting. In this case, they have to first consider the specifications needed for a technology to work (who will be using it, what the expectations are, etc.) and, secondly, what the relationship is between the study of language and designing a product which is functional.

Fun homework questions In the homeworks, students are often instructed to use a technology on the internet, or in some way to take the material presented in class a step farther. Additionally, most homework assignments had at least one lighter question which allowed students to be more creative in their responses while at the same time reinforcing the material.

For example, instructors have asked students to send them spam, and the most spam-worthy message won a prize. Other homework questions have included sketching out what it would take to convert an ELIZA system into a hostage negotiator—and what the potential dangers are in such a use. Although some students put down minimal answers, many students offer pages of detailed suggestions to answer such a question. This gives students a taste of the creativity involved in designing new technology without having to deal with the technicalities.

6 Challenges for the course

Despite the positive response, there are several aspects to the course which have needed improvement and continue to do so. Teaching to a diverse audience of interests and capabilities presents obstacles which are not easily overcome. To that end, here we will review aspects of the course which students did not generally enjoy and which we are in the process of adapting to better suit our purposes and our students' needs.

6.1 Topics they do not relate to

For such a range of students, there is the difficulty of presenting abstract concepts. Although we try to relate everything to something which students actually use or could readily use, we sometimes include topics from computational linguistics that make one better able to think logically in general and which we feel will be of future use for our students. One such topic is that of regular expressions, in the context of searching for text in a document or corpus. As most students only experience searching as part of what they do on the web, and no web search engine (to the best of our knowledge) currently supports regular expression searching, students often wonder what the point of the topic is. In making most topics applicable to everyday life, we had raised expect. In this particular case, students seemed to accept regular expressions more once it they saw that Microsoft Word has something roughly analogous.

Another difficulty that presented itself for a subset of the students was that of using foreign language text to assist in teaching machine translation and computer-aided language learning. Every example was provided with an English word-by-word gloss, as well as a paraphrase, yet the examples can still be difficult to understand without a basic appreciation for the relevant languages. If the students know Spanish, the example is in Spanish and the instructor has a decent Spanish accent, things can go well. But students tend to blame difficulties in the machine translation homework on not knowing the languages used in the examples. Understanding the distinction between different kinds of machine translation systems requires some ability to grasp how languages can differ, so we certainly must (unless we use proxies like fairytale English) present some foreign material, but we are in dire need of means to do this as gently as possible

6.2 Math they do not understand

While some of the more difficult mathematical concepts were eventually understood, others continued to frustrate students. The already mentioned regular expressions, for example, caused trouble. Firstly, even if you do understand them, they are not necessarily life-enhancing, unless you are geeky enough to write your papers in a text editor that properly supports them. Secondly, and more importantly, many students saw them as unnecessarily abstract and complex. For instance, some students were simply unable to understand the notion that the Kleene star is to be interpreted as an operator rather than as a special character occurring in place of any string.

Even though we thought we had calibrated our expectations to respect the fact that our students knew no math beyond high school, the amount that they had retained from high school was often less than we expected. For example, many students behaved exactly as if they had never seen Venn diagrams before, so time had to be taken away from the main material in order to explain them. Likewise, figuring out how to calculate probabilities for a bag of words model of statistical machine translation required a step-by-step explanation of where each number comes from. A midterm question on Bayesian spam filtering needed the same treatment, revealing that even good students may have significant difficulties in deploying the high school math knowledge they almost certainly possess.

6.3 Technology which did not work

Most assignments required students to use the internet or the phone in some capacity, usually to try out a demo. With such tasks, there is always the danger that the technology will not work. For example, during the first quarter the course was taught, students were asked to call the CMU Communicator system and interact with it, to get a feel for what it is like to interact with a computer. As it turns out, halfway through the week the assignment was due, the system was down, and thus some students could not finish the exercise. Following this episode, homework questions now come with alternate questions. In this case, if the system is down, the first alternate is to listen to a pre-recorded conversation to see how the Communicator works. Since some students are unable to listen to sounds in the campus computer labs, the second alternate is to read a transcript.

Likewise, students were instructed to view the page source code for ELIZA. However, some campus computer labs at OSU do not allow students to view the source of a webpage. In response to this, current versions of the assignment have a separate webpage with the source code written out as plain text, so all students can view it.

One final note is that students have often complained of weblinks failing to work, but this "failure" is most often due to students mistyping the link provided in the homework. Providing links directly on the course webpage or including them in the web- or pdf-versions of the homework sheets is the simplest solution for this problem.

7 Summary and Outlook

We have described the course Language and Computers (Linguistics 384), a general introduction to computational linguistics currently being taught at OSU. While there are clear lessons to be learned for developing similar courses at other universities, there are also more general points to be made. In courses which assume some CS background, for instance, it is still likely the case that students will want to see some practical use of what they are doing and learning.

There are several ways in which this course can continue to be improved. The most pressing priority is to develop a course packet and possibly a textbook. Right now, students rely only on the instructor's handouts, and we would like to provide a more in-depth and cohesive source of material. Along with this, we want to develop a wider range of readings for students (e.g. Dickinson, to appear) to provide students with a wider variety of perspectives and explanations for difficult concepts.

To address the wide range of interests and capabilities of the students taking this course as a general education requirement, it would be good to tailor some of the sections to audiences with specific backgrounds—but given the lack of a dedicated free time slot for all students of a particular major, etc., it is unclear whether this is feasible in practice.

We are doing reasonably well in integrating mathematical thinking into the course, but we would like to give students more experience of thinking about algorithms. Introducing a basic form of pseudocode might go some way towards achieving this, provided we can find a motivating linguistic example that is both simple enough to grasp and complex enough to justify the overhead of introducing a new topic. Further developments might assist us in developing a course between Linguistics 384 and Linguistics 684, our graduate-level computational linguistics course, as we currently have few options for advanced undergraduates.

Acknowledgements We would like to thank the instructors of Language and Computers for their discussions and insights into making it a better course: Stacey Bailey, Anna Feldman, Xiaofei Lu, Crystal Nakatsu, and Jihyun Park. We are also grateful to the two ACL-TNLP reviewers for their detailed and helpful comments.

References

Markus Dickinson, to appear. Writers' Aids. In Keith Brown (ed.), *Encyclopedia of Language and Linguistics. Second Edition*, Elsevier, Oxford.

Malcolm Gladwell, 2001. The Social Life of Paper. *New Yorker*. available from `http://www.gladwell.com/archive.html`.

Lillian Lee, 2002. A non-programming introduction to computer science via NLP, IR, and AI. In *ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*. pp. 32–37.

A.M. Turing, 1950. Computing Machinery and Intelligence. *Mind*, 59(236):433–460.

A Core-Tools Statistical NLP Course

Dan Klein
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
klein@cs.berkeley.edu

Abstract

In the fall term of 2004, I taught a new statistical NLP course focusing on core tools and machine-learning algorithms. The course work was organized around four substantial programming assignments in which the students implemented the important parts of several core tools, including language models (for speech reranking), a maximum entropy classifier, a part-of-speech tagger, a PCFG parser, and a word-alignment system. Using provided scaffolding, students built realistic tools with nearly state-of-the-art performance in most cases. This paper briefly outlines the coverage of the course, the scope of the assignments, and some of the lessons learned in teaching the course in this way.

1 Introduction

In the fall term of 2004, I taught a new statistical NLP course at UC Berkeley which covered the central tools and machine-learning approaches of NLP. My goal in formulating this course was to create a syllabus and assignment set to teach in a relatively short time the important aspects, both practical and theoretical, of what took me years of building research tools to internalize. The result was a rather hard course with a high workload. Although the course evaluations were very positive, and several of the students who completed the course were able to jump right into research projects in my group, there's no question that the broad accessibility of the course, especially for non-CS students, was limited.

As with any NLP course, there were several fundamental choice points. First, it's not possible to cover both core tools and end-to-end applications in detail in a single term. Since Marti Hearst was teaching an applied NLP course during the same term, I chose to cover tools and algorithms almost exclusively (see figure 1 for a syllabus). The second choice point was whether to organize the course primarily around linguistic topics or primarily around statistical methods. I chose to follow linguistic topics because that order seemed much easier to motivate to the students (comments on this choice in section 3). The final fundamental choice I made in deciding how to target this class was to require both substantial coding and substantial math. This choice narrowed the audience of the class, but allowed the students to build realistic systems which were not just toy implementations.

I feel that the most successful aspect of this course was the set of assignments, so the largest section below will be devoted to describing them. If other researchers are interested in using any of my materials, they are encouraged to contact me or visit my web page (http://www.cs.berkeley.edu/~klein).

2 Audience

The audience of the class began as a mix of CS PhD students (mostly AI but some systems students), some linguistics graduate students, and a few advanced CS undergrads. What became apparent after the first homework assignment (see section 4.2) was that while the CS students could at least muddle through the course with weak (or absent) linguistics backgrounds, the linguistics students were unable to acquire the math and programming skills quickly enough to keep up. I have no good ideas about how to address this issue. Moreover, even among the CS students, some of the systems students had trouble with the math and some of the AI/theory students had issues with coding scalable solutions. The course was certainly not optimized for broad accessibility, but the approximately 80% of students who stuck it out did what I considered to be extremely impressive work. For example, one student built a language model which took the mass reserved for new words and distributed it according to a character n-gram model. Another student invented a non-iterative word alignment heuristic which outperformed IBM model 4 on small and medium training corpora. A third student built a maxent part-of-speech tagger with a per-word accuracy of 96.7%, certainly in the state-of-the-art range.

3 Topics

The topics covered in the course are shown in figure 1. The first week of the course was essentially a history lesson about symbolic approaches NLP, both to show their strengths (a full, unified pipeline including predicate logic semantic interpretations, while we still don't have a good notion of probabilistic interpretation) and their weaknesses (many interpretations arise from just a few rules, ambiguity poorly handled). From there, I discussed statistical approaches to problems of increasing complexity, spending a large amount of time on tree and sequence models.

As mentioned above, I organized the lectures around linguistic topics rather than mathematical methods. However, given the degree to which the course focused on such foundational methods, this order was perhaps a mistake. For example, it meant that simple word alignment models like IBM models 1 and 2 (Brown et al., 1990) and the HMM model (Vogel et al., 1996) came many weeks after HMMs were introduced in the context of part-of-speech tagging. I also separated unsupervised learning into its own sub-sequence, where I now wish I had presented the unsupervised approaches to each task along with the supervised ones.

I assigned readings from Jurafsky and Martin (2000) and Manning and Schütze (1999) for the first half of the course, but the second half was almost entirely based on papers from the research literature. This reflected both increasing sophistication on the part of the students and insufficient coverage of the latter topics in the textbooks.

4 Assignments

The key component which characterized this course was the assignments. Each assignment is described below. They are available for use by other instructors. While licensing issues with the data make it impossible to put the entirety of the assignment materials on the web, some materials will be linked from `http://www.cs.berkeley.edu/~klein`, and the rest can be obtained by emailing me.

4.1 Assignment Principles

The assignments were all in Java. In all cases, I supplied a large amount of scaffolding code which read in the appropriate data files, constructed a placeholder baseline system, and tested that baseline. The students therefore always began with a running end-to-end pipeline, using standard corpora, evaluated in standard ways. They then swapped out the baseline placeholder for increasingly sophisticated implementations. When possible, assignments also had a toy "miniTest" mode where rather than reading in real corpora, a small toy corpus was loaded to facilitate debugging. Assignments were graded entirely on the basis of write-ups.

4.2 Assignment 1: Language Modeling

In the first assignment, students built n-gram language models using WSJ data. Their language models were evaluated in three ways by

Topics	Techniques	Lectures
Classical NLP	Chart Parsing, Semantic Interpretation	2
Speech and Language Modeling	Smoothing	2
Text Categorization	Naive-Bayes Models	1
Word-Sense Disambiguation	Maximum Entropy Models	1
Part-of-Speech Tagging	HMMs and MEMMs	1
Part-of-Speech Tagging	CRFs	1
Statistical Parsing	PCFGs	1
Statistical Parsing	Inference for PCFGs	1
Statistical Parsing	Grammar Representations	1
Statistical Parsing	Lexicalized Dependency Models	1
Statistical Parsing	Other Parsing Models	1
Semantic Representation		2
Information Extraction		1
Coreference		1
Machine Translation	Word-to-Word Alignment Models	1
Machine Translation	Decoding Word-to-Word Models	1
Machine Translation	Syntactic Translation Models	1
Unsupervised Learning	Document Clustering	1
Unsupervised Learning	Word-Level Clustering	1
Unsupervised Learning	Grammar Induction	2
Question Answering		1
Document Summarization		1

Figure 1: Topics Covered. Each lecture was 80 minutes.

the support harness. First, perplexity on held-out WSJ text was calculated. In this evaluation, reserving the correct mass for unknown words was important. Second, their language models were used to rescore n-best speech lists (supplied by Brian Roark, see Roark (2001)). Finally, random sentences were generatively sampled from their models, giving students concrete feedback on how their models did (or did not) capture information about English. The support code intially provided an unsmoothed unigram model to get students started. They were then asked to build several more complex language models, including at least one higher-order interpolated model, and at least one model using Good-Turing or held-out smoothing. Beyond these requirements, students were encouraged to acheive the best possible word error rate and perplexity figures by whatever means they chose.[1] They were also asked to identify ways in which their language models missed important trends of English and to suggest solutions.

As a second part to assignment 1, students trained class-conditional n-gram models (at the character level) to do the proper name identification task from Smarr and Manning (2002) (whose data we used). In this task, proper name strings are to be mapped to one of {DRUG, COMPANY, MOVIE, PERSON, LOCATION}. This turns out to be a fairly easy task since the different categories have markedly different character distributions.[2] In the future, I will move this part of assignment 1 and the matching part of assignment 2 into a new, joint assignment.

4.3 Assignment 2: Maximum Entropy / POS Tagging

In assignment 2, students first built a general maximum entropy model for multiclass classification. The support code provided a crippled maxent classifier which always returned the uniform distribution over labels (by ignoring the features of the input datum). Students replaced the crippled bits and got a correct classifier run-

[1]After each assignment, I presented in class an honors list, consisting of the students who won on any measure or who had simply built something clever. I initially worried about how these honors announcements would be received, but students really seemed to enjoy hearing what their peers were doing, and most students made the honors list at some point in the term.

[2]This assignment could equally well have been done as a language identification task, but the proper name data was convenient and led to fun error analysis, since in good systems the errors are mostly places named after people, movies with place names as titles, and so on.

ning, first on a small toy problem and then on the proper-name identification problem from assignment 1. The support code provided optimization code (an L-BFGS optimizer) and feature indexing machinery, so students only wrote code to calculate the maxent objective function and its derivatives.

The original intention of assignment 2 was that students then use this maxent classifier as a building block of a maxent part-of-speech tagger like that of Ratnaparkhi (1996). The support code supplied a most-frequent-tag baseline tagger and a greedy lattice decoder. The students first improved the local scoring function (keeping the greedy decoder) using either an HMM or maxent model for each timeslice. Once this was complete, they upgraded the greedy decoder to a Viterbi decoder. Since students were, in practice, generally only willing to wait about 20 minutes for an experiment to run, most chose to discard their maxent classifiers and build generative HMM taggers. About half of the students' final taggers exceeded 96% per-word tagging accuracy, which I found very impressive. Students were only required to build a trigram tagger of some kind. However, many chose to have smoothed HMMs with complex emission models like Brants (2000), while others built maxent taggers.

Because of the slowness of maxent taggers' training, I will just ask students to build HMM taggers next time. Moreover, with the relation between the two parts of this assignment gone, I will separate out the proper-name classification part into its own assignment.

4.4 Assignment 3: Parsing

In assignment 3, students wrote a probabilistic chart parser. The support code read in and normalized Penn Treebank trees using the standard data splits, handled binarization of n-ary rules, and calculated ParsEval numbers over the development or test sets. A baseline left-branching parser was provided. Students wrote an agenda-based uniform-cost parser essentially from scratch. Once the parser parsed correctly with the supplied treebank grammar, students experimented with horizontal and vertical markovization (see Klein and Manning (2003)) to improve parsing accuracy. Students were then free to experiment with speed-ups to the parser, more complex annotation schemes, and so on. Most students' parsers ran at reasonable speeds (around a minute for 40 word sentences) and got final F_1 measures over 82%, which is substantially higher than an unannotated treebank grammar will produce. While this assignment would appear to be more work than the others, it actually got the least overload-related complaints of all the assignments.

In the future, I may instead have students implement an array-based CKY parser (Kasami, 1965), since a better understanding of CKY would have been more useful than knowing about agenda-based methods for later parts of the course. Moreover, several students wanted to experiment with induction methods which required summing parsers instead of Viterbi parsers.

4.5 Assignment 4: Word Alignment

In assignment 4, students built word alignment systems using the Canadian Hansards training data and evaluation alignments from the 2003 (and now 2005) shared task in the NAACL workshop on parallel texts. The support code provided a monotone baseline aligner and evaluation/display code which graphically printed gold alignments superimposed over guessed alignments. Students first built a heuristic aligner (Dice, mutual information-based, or whatever they could invent) and then built IBM model 1 and 2 aligners. They then had a choice of either scaling up the system to learn from larger training sets or implementing the HMM alignment model.

4.6 Assignment Observations

For all the assignments, I stressed that the students should spend a substantial amount of time doing error analysis. However, most didn't, except for in assignment 2, where the support code printed out every error their taggers made, by default. For this assignment, students actually provided very good error analysis. In the future, I will increase the amount of verbose error output to encourage better error analysis for the other assignments – it seemed like students were reluctant to write code to display errors, but were happy to look at errors as they scrolled by.[3]

A very important question raised by an anonymous reviewer was how effectively implementing tried-and-true methods feeds into new research. For students who will not be doing NLP research but want to know how the basic methods work (realistically, this is most of the audience), the experience of having implemented several "classic" approaches to core tools is certainly appropriate. However, even for students who intend to do NLP research, this hands-on tour of established methods has already shown itself to be very valuable. These students can pick up any paper on any of these tasks, and they have a very concrete idea about what the data sets look like, why people do things they way they do, and what kinds of error types and rates one can expect from a given tool. That's experience that can take a long time to acquire otherwise – it certainly took me a while. Moreover, I've had several students from the class start research projects with me, and, in each case, those projects have been in some way bridged by the course assignments. This methodology also means that all of the students working with me have a shared implementation background, which has facilitated ad hoc collaborations on research projects.

5 Conclusions

There are certainly changes I will make when I teach this course again this fall. I will likely shuffle the topics around so that word alignment comes earlier (closer to HMMs for tagging) and I will likely teach dynamic programming solutions to parsing and tagging in more depth than graph-search based methods. Some students needed remedial linguistics sections and other students needed remedial math sections, and I would hold more such sessions, and earlier in the term. However, I will certainly keep the substantial implementation component of the course, partially in response to very positive student feedback on the assignments, partially from my own reaction to the high quality of student work on those assignments, and partially from how easily students with so much hands-on experience seem to be able to jump into NLP research.

[3]There was also verbose error reporting for assignment 4, which displayed each sentence's guessed and gold alignments in a grid, but since most students didn't speak French, this didn't have the same effect.

References

Thorsten Brants. 2000. TnT – a statistical part-of-speech tagger. In *ANLP 6*, pages 224–231.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.

Dan Jurafsky and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition.* Prentice Hall, Englewood Cliffs, NJ.

T. Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *ACL 41*, pages 423–430.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge, Massachusetts.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *EMNLP 1*, pages 133–142.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27:249–276.

Joseph Smarr and Christopher D. Manning. 2002. Classifying unknown proper noun phrases without context. Technical report, Stanford University.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-based word alignment in statistical translation. In *COLING 16*, pages 836–841.

Web-based Interfaces for Natural Language Processing Tools

Marc Light[†] and **Robert Arens**[*] **and Xin Lu**[*]
[†]Linguistics Department
[†]School of Library and Information Science
[*†]Computer Science Department
University of Iowa
Iowa, USA 52242
{marc-light,robert-arens,xin-lu}@uiowa.edu

Abstract

We have built web interfaces to a number of Natural Language Processing technologies. These interfaces allow students to experiment with different inputs and view corresponding output and inner workings of the systems. When possible, the interfaces also enable the student to modify the knowledge bases of the systems and view the resulting change in behavior. Such interfaces are important because they allow students **without** computer science background to learn by doing. Web interfaces also sidestep issues of platform dependency in software packages, available computer lab times, etc. We discuss our basic approach and lessons learned.

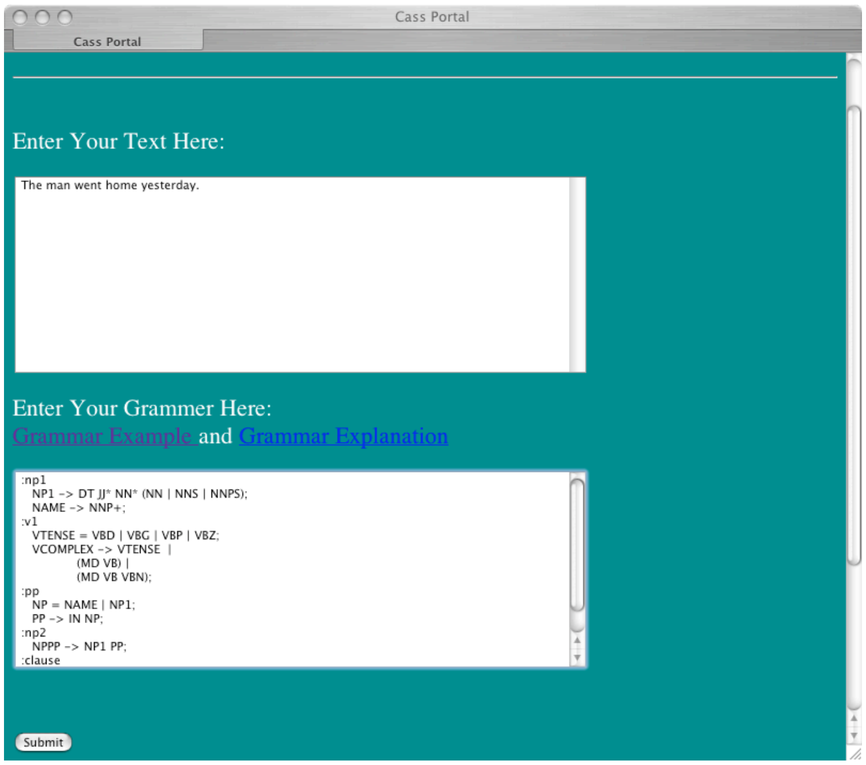
1 Introduction

The Problem: Natural language processing (NLP) technology is relevant to non-computer scientists: our classes are populated by students from neuroscience, speech pathology, linguistics, teaching of foreign languages, health informatics, etc. To effectively use NLP technology, it is helpful understand, at some level, how it works. Hands-on experimentation is an effective method for gaining such understanding. **Unfortunately**, to be able to experiment, non-computer scientists often need to acquire some programming skills and knowledge of the Unix operating system. This can be time consuming and tedious and can distract students from their central goal of understanding how a technology works and how best to employ it for their interests.

In addition, getting a technology to run on a set lab machines can be problematic: the programs may be developed for a different platform, e.g., a program was developed for Linux but the lab machines run MSWindows. Another hurdle is that machine administrators are often loath to install applications that they perceive as non-standard. Finally, lab times can be restrictive and thus it is preferable to enable students to use computers to which they have easy access.

Our Solution: We built web interfaces to many core NLP modules. These interfaces not only allow students to use a technology but also allow students to modify and extend the technology. This enables experimentation. We used server-side scripting languages to build such web interfaces. These programs take input from a web browser, feed it to the technology in question, gather the output from the technology and send it back to the browser for display to the student. Access to web browsers is nearly ubiquitous and thus the issue of lab access is side-stepped. Finally, the core technology need only run on the web server platform. Many instructors have access to web servers running on different platforms and, in general, administering a web server is easier than maintaining lab machines.

An Example: Finite state transduction is a core NLP technology and one that students need to understand. The Cass partial parsing system (Abney, 1997) makes use of a cascade of FSTs. To use this system, a student creates a grammar. This grammar is compiled and then applied to sentences provided



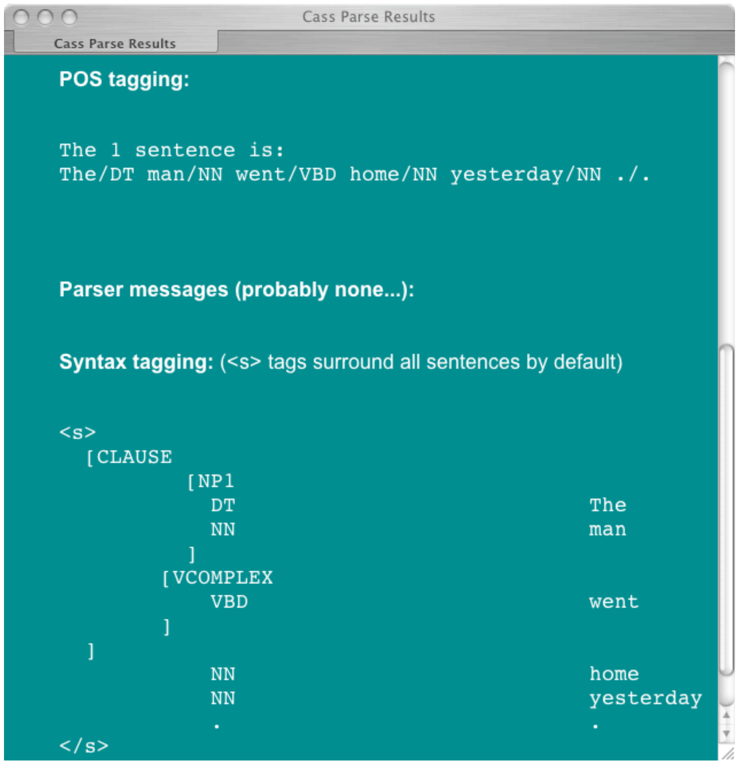
Enter Your Text Here:

```
The man went home yesterday.
```

Enter Your Grammer Here:
Grammar Example and Grammar Explanation

```
:np1
  NP1 -> DT JJ* NN* (NN | NNS | NNPS);
  NAME -> NNP+;
:v1
  VTENSE = VBD | VBG | VBP | VBZ;
  VCOMPLEX -> VTENSE  |
          (MD VB) |
          (MD VB VBN);
:pp
  NP = NAME | NP1;
  PP -> IN NP;
:np2
  NPPP -> NP1 PP;
:clause
```

Figure 1: Web interface to Cass



POS tagging:

```
The 1 sentence is:
The/DT man/NN went/VBD home/NN yesterday/NN ./.
```

Parser messages (probably none...):

Syntax tagging: (<s> tags surround all sentences by default)

```
<s>
  [CLAUSE
          [NP1
            DT                              The
            NN                              man
          ]
        [VCOMPLEX
            VBD                             went
        ]
  ]
            NN                              home
            NN                              yesterday
            .                               .
</s>
```

Figure 2: Cass Output

by the student. Prior to our work, the only interface to Cass involved the Unix command line shell. Figure 3 shows an example session with the command line interface. It exemplifies the sort of interface that users must master in order to work with current human language technology.

```
emacs input.txt &
emacs grammar.txt &
source /usr/local/bin/setupEnv
reg gram.txt
Montytagger.py inTagged input.txt
cat inTagged |
wordSlashTagInput.pl |
cass -v -g gram.txt.fsc > cassOut
less cassOut
```

Figure 3: Cass Command Line Interface

A web-based interface hides many of the details, see Figure 1 and Figure 2. For example, the use of an ASCII-based text editor such as `emacs` become unnecessary. In addition, the student does not need to remembering flags such as `-v -g` and does not need to know how to use Unix pipes, |, and output redirection, >. None of this knowledge is terribly difficult but the amount accumulates quickly and such information does *not* help the student understand how Cass works.

2 What we have built

To date, we have built web interfaces to nine NLP-related technologies:

- the Cass parser (Abney, 1997),
- the MontyTagger Brill-style part-of-speech tagger (Liu, 2004),
- the NLTK statistical part-of-speech tagger,
- a NLTK context-free grammar parser (Loper and Bird, 2002),
- the Gsearch context-free grammar parser (Corley et al., 2001),
- the SenseRelate word sense disambiguation system (Pedersen et al., 2005),
- a Perl Regular expression evaluator,
- a linguistic feature annotator,
- and a decision tree classifier (Witten and Frank, 1999).

These interfaces have been used in an introduction to computational linguistics course and an introduction to creating and using corpora course. Prior to the interface construction, no hands-on lab assignments were given; instead all assignments were pencil and paper. The NLP technologies listed above were chosen because they fit into the material of the course and because of their availability.

2.1 Allowing the student to process input

The simplest type of interface allows students to provide input and displays corresponding output. All the interfaces above provide this ability. They all start with HTML forms to collect input. In the simplest case, PHP scripts process the forms, placing input into files and then system calls are made to run the NLP technology. Finally, output files are wrapped in HTML and displayed to the user. The basic PHP program remains largely unchanged from one NLP technology to the next. In most cases, it suffices to use the server file system to pass data back and forth to the NLP program — PHP provides primitives for creating and removing unique temporary files. In only one case was it necessary to use a semaphore on a hard-coded filename. We also experimented with Java server pages and Perl CGI scripts instead of PHP.

2.2 Allowing the student to modify knowledge resources

The web interfaces to the Cass parser, Gsearch, and MontyTagger allow the student to provide their corresponding knowledge base. For Cass and Gsearch, an additional text box is provided for the grammars they require. The rule sequence and lexicon that the MontyTagger uses can be large and thus unwieldy for a `textarea` form input element. We solved the problem by preloading the `textarea`s with a "standard" rule sequence and lexicon which the student can then modify. We also provided the ability to upload the rule sequences and lexicon as files. One problem with the file upload method is that it assume that the students can generate ASCII-only files with the appropriate line break character. This assumption is often false.

An additional problem with allowing students to modify knowledge resources is providing useful feedback when these student-provided resources contain syntax or other types of errors. At this point we simply capture the `stderr` output of the program and display it.

Finally, with some systems such as Spew (Schwartz, 1999), and The Dada Engine (Bulhak, 1996), allowing web-based specification of knowledge bases amounts to allowing the student to execute arbitrary code on the server machine, an obvious security problem.

2.3 Allowing the student to examine internal system processing

Displaying system output with a web interface is relatively easy; however, showing the internal workings of a system is more challenging with a web interface. At this point, we have only displayed traces of steps of an algorithm. For example, the NLTK context-free grammar parser interface provides a trace of the steps of the parsing algorithm. One possible solution would be to generate Flash code to animate a system's processing.

2.4 Availability

The web pages are currently available at que.info-science.uiowa.edu/˜light/classes/compLing/ However, it is not our intent to provide server cycles for the community but rather to provide the PHP scripts open source so that others can run the interfaces on their own servers. An instructor at another university has already made use of our code.

3 Lessons learned

- PHP is easier to work with than Java Server Pages and CGI scripts;
- requiring users to paste input into text boxes is superior to allowing user to upload files (for security reasons and because it is easier to control the character encoding used);
- getting debugging information back to the student is very important;
- security is an issue since one is allowing users to initiate computationally intensive processes;
- it is still possible for students to claim the interface does not work for them (even though we used no client-side scripting).
- Peer learning is less likely than in a lab setting; however, we provided a web forum and this seems to alleviated the problem somewhat.

4 Summary

At the University of Iowa, many students, who want to learn about natural language processing, do not have the requisite Unix and programming skills to do labs using command line interfaces. In addition, our lab machines run MSWindows, the instructors do not administer the machines, and there are restrictive lab hours. Thus, until recently assignments consisted of pencil-and-paper problems. We have built web-based interfaces to a number of NLP modules that allow students to use, modify, and learn.

References

Steven Abney. 1997. Partial parsing via finite-state cascades. *Natural Language Engineering*, 2(4).

Andrew Bulhak. 1996. The dada engine. http://dev.null.org/dadaengine/.

S. Corley, M. Corley, F. Keller, M. Crocker, and S. Trewin. 2001. Finding Syntactic Structure in Unparsed Corpora: The Gsearch Corpus Query System. *Computers and the Humanities*, 35:81–94.

Hugo Liu. 2004. Montylingua: An end-to-end natural language processor with common sense. homepage.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *Proc. of the ACL-02 Workshop on Effective Tools and Methods for Teaching Natural Language Processing and Computational Linguistics*.

Ted Pedersen, Satanjeev Banerjee, and Siddharth Patwardhan. 2005. Maximizing Semantic Relatedness to Perform Word Sense Disambiguation. Supercomputing institute research report umsi 2005/25, University of Minnesota.

Randal Schwartz. 1999. Random sentence generator. *Linux Magazine*, September.

Ian H. Witten and Eibe Frank. 1999. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.

Making Hidden Markov Models More Transparent

Nashira Richard Lincoln[*] and **Marc Light**[†]
[*†]Linguistics Department
[†]School of Library and Information Science
[†]Computer Science Department
University of Iowa
Iowa, USA 52242
{nashira-lincoln, marc-light}@uiowa.edu

Abstract

Understanding the decoding algorithm for hidden Markov models is a difficult task for many students. A comprehensive understanding is difficult to gain from static state transition diagrams and tables of observation production probabilities. We have built a number of visualizations depicting a hidden Markov model for part-of-speech tagging and the operation of the Viterbi algorithm. The visualizations are designed to help students grasp the operation of the HMM. In addition, we have found that the displays are useful as debugging tools for experienced researchers.

1 Introduction

Hidden Markov Models (HMMs) are an important part of the natural language processing toolkit and are often one of the first stochastic generation models that students[1] encounter. The corresponding Viterbi algorithm is also often the first example of dynamic programming that students encounter. Thus, HMMs provide an opportunity to start students on the correct path of understanding stochastic models, **not** simply treating them as black boxes. Unfortunately, static state transition diagrams, tables of probability values, and lattice diagrams are not enough for many students. They have a general idea of how a HMM works but often have common misconceptions. For example, we have found that students often believe that as the Viterbi algorithm calculates joint state sequence observation sequence probabilities, the best state sequence so far is always a prefix of global best path. This is of course false. Working a long example to show this is very tedious and thus text books seldom provide such examples.

Even for practitioners, HMMs are often opaque in that the cause of a mis-tagging error is often left uncharacterized. A display would be helpful to pinpoint why an HMM chose an incorrect state sequence instead of the correct one.

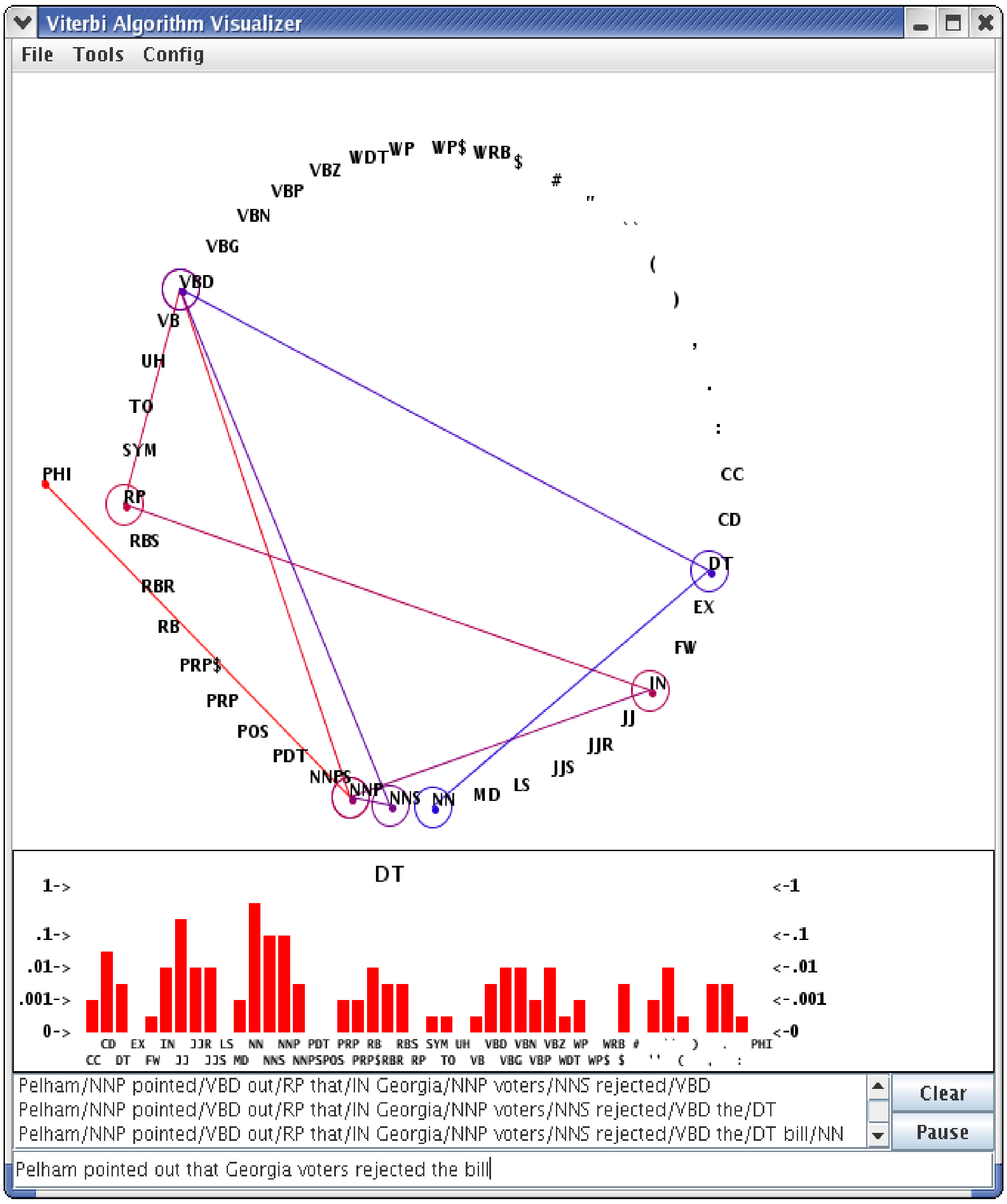
Below we describe two displays that attempt to remedy the above mentioned problems and we discuss a Java implementation of these displays in the context of a part-of-speech tagging HMM (Kupiec, 1992). The system is freely available and has an XML model specification that allows models calculated by other methods to be viewed. (A standard maximum likelihood estimation was implemented and can be used to create models from tagged data. A model is also provided.)

2 Displays

Figure 1 shows a snapshot of our first display. It contains three kinds of information: most likely path for input, transition probabilities, and history of most likely prefixes for each observation index in the Viterbi lattice. The user can input text at the bottom of the display, e.g., *Pelham pointed out that Georgia voters rejected the bill*. The system then runs Viterbi and animates the search through all possible state sequences and displays the best state sequence prefix as it works its way through the observation

[1]The Introduction to Computational Linguistics course at the University of Iowa has no prerequisites, and over half the students are not CS majors.

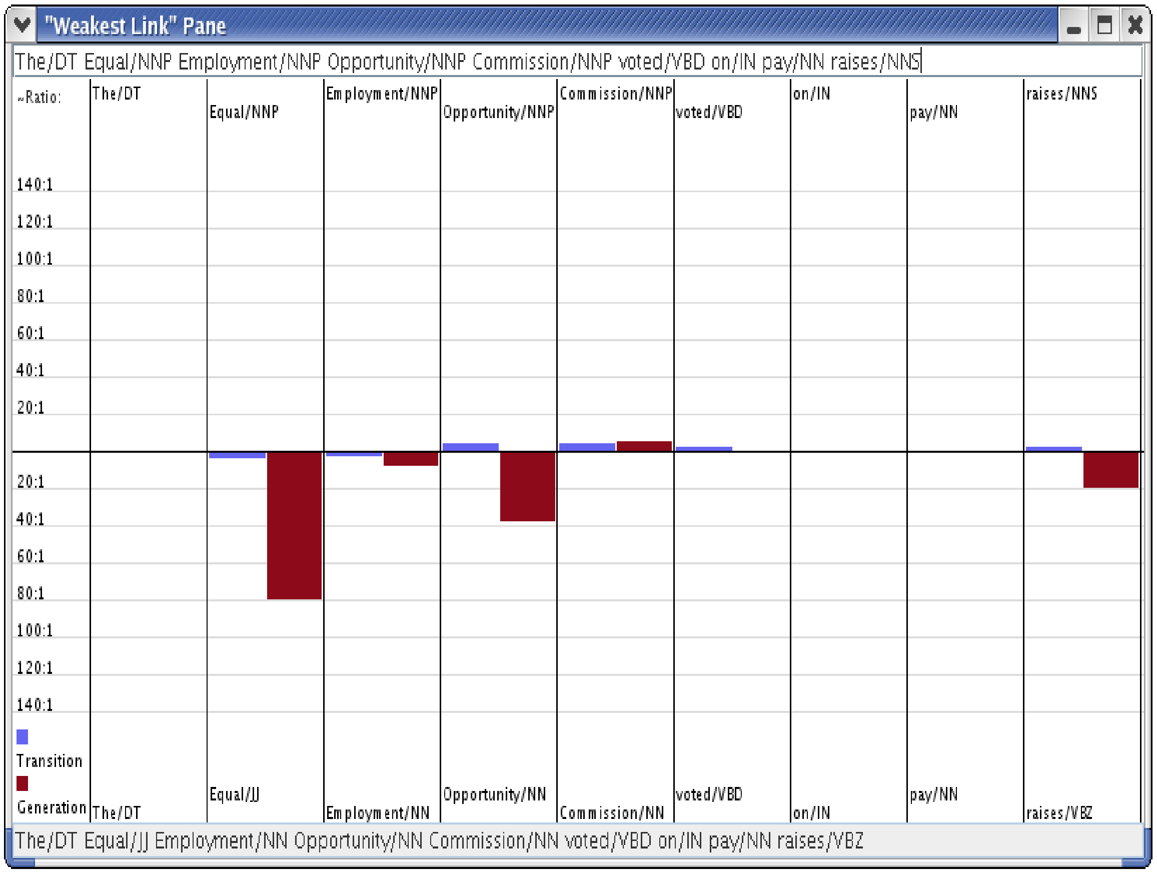
Proceedings of the Second ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL, pages 32–36,
Ann Arbor, June 2005. ©2005 Association for Computational Linguistics


Viterbi Algorithm Visualizer
File Tools Config
VBZ WDT WP WP$ WRB $ # '' `` ( ) , . : CC CD DT EX FW IN JJ JJR JJS LS MD NN NNS NNP NNPS PDT POS PRP PRP$ RB RBR RBS RP PHI SYM TO UH VB VBD VBG VBN VBP
DT
1-> .1-> .01-> .001-> 0->
<-1 <-.1 <-.01 <-.001 <-0
CD EX IN JJR LS NN NNP PDT PRP RB RBS SYM UH VBD VBN VBZ WP WRB # ``) . PHI
CC DT FW JJ JJS MD NNS NNPSPOS PRP$RBR RP TO VB VBG VBP WDT WP$ $ '' (, :
Clear
Pause

```
Pelham/NNP pointed/VBD out/RP that/IN Georgia/NNP voters/NNS rejected/VBD
Pelham/NNP pointed/VBD out/RP that/IN Georgia/NNP voters/NNS rejected/VBD the/DT
Pelham/NNP pointed/VBD out/RP that/IN Georgia/NNP voters/NNS rejected/VBD the/DT bill/NN
```

```
Pelham pointed out that Georgia voters rejected the bill
```

Figure 1: The system's main display. **Top pane**: shows the state space and animates the derivation of the most likely path for "Pelman pointed out that Georgia voters ..."; **Middle pane**: a mouse-over-triggered bar graph of out transition probabilities for a state; **Bottom pane**: a history of most likely prefixes for each observation index in the Viterbi lattice. Below the panes is the input text field.


The/DT Equal/NNP Employment/NNP Opportunity/NNP Commission/NNP voted/VBD on/IN pay/NN raises/NNS

The/DT Equal/JJ Employment/NN Opportunity/NN Commission/NN voted/VBD on/IN pay/NN raises/VBZ

Figure 2: Contrast display: The user enters a sequence on the top text field and presses enter, the sequence is tagged and displayed in both the top and bottom text fields. Finally, the user changes any incorrect tags in the top text field and presses enter and the probability ratio bars are then displayed.

sequence from left to right (these are lines connecting the states in Figure 1). At any point, the student can mouse-over a state to see probabilities for transitions out of that state (this is the bar graph in Figure 1). Finally, the history of most likely prefixes is displayed (this history appears below the bar graph in Figure 1). We mentioned that students often falsely believe that the most likely prefix is extended monotonically. By seeing the path through the states reconfigure itself in the middle of the observation sequence and by looking at the prefix history, a student has a good chance of dispelling the false belief of monotonicity.

The second display allows the user to contrast two state sequences for the same observation sequence. See Figure 2. For each contrasting state pairs, it shows the ratio of the corresponding transition to each state and it shows the ratio of the generation of the observation conditioned on each state. For example, in Figure 2 the transition DT→JJ is less likely than DT→NNP. The real culprit is generation probability P(*Equal*|JJ) which is almost 7 times larger than P(*Equal*|NNP). Later in the sequence we see a similar problem with generating *opportunity* from a NNP state. These generation probabilities seem to drown out any gains made by the likelihood of NNP runs.

To use this display, the user types in a sentence in the box above the graph and presses enter. The HMM is used to tag the input. The user then modifies (e.g., corrects) the tag sequence and presses enter and the ratio bars then appear.

Let us consider another example: in Figure 2, the mis-tagging of *raises* as a verb instead of a noun at the end of the sentence. The display shows us that although NN→NNS is more likely than NN→VBZ, the generation probability for *raises* as a verb is over twice as high as a noun. (If this pattern of mis-taggings caused by high generation probability ratios was found repeatedly, we might consider smoothing these distributions more aggressively.)

3 Implementation

The HMM part-of-speech tagging model and corresponding Viterbi algorithm were implemented based on their description in the updated version, `http://www.cs.colorado.edu/˜martin/SLP/updated.html`, of chapter 8 of (Jurafsky and Martin, 2000). A model was trained using Maximum Likelihood from the UPenn Treebank (Marcus et al., 1993). The input model file is encoded using XML and thus models built by other systems can be read in and displayed.

The system is implemented in Java and requires 1.4 or higher to run. It has been tested on Linux and Apple operating systems. We will release it under a standard open source license.

4 Summary and future work

Students (and researchers) need to understand HMMs. We have built a display that allow users to probe different aspects of an HMM and watch Viterbi in action. In addition, our system provides a display that allows users to contrast state sequence probabilities. To drive these displays, we have built a standard HMM system including parameter estimating and decoding and provide a part-of-speech model trained on UPenn Treebank data. The system can also read in models constructed by other systems.

This system was built during this year's offering of *Introduction to Computational Linguistics* at the University of Iowa. In the Spring of 2006 it will be deployed in the classroom for the first time. We plan on giving a demonstration of the system during a lecture on HMMs and part-of-speech tagging. A related problem set using the system will be assigned. The students will be given several mis-tagged sentences and asked to analyze the errors and report on precisely why they occurred. A survey will be administered at the end and improvements will be made to the system based on the feedback provided.

In the future we plan to implement Good-Turing smoothing and a method for dealing with unknown words. We also plan to provide an additional display that shows the traditional Viterbi lattice figure, i.e., observations listed left-to-right, possible states listed from top-to-bottom, and lines from left-to-right connecting states at observation index *i* with the previous states, *i-1*, that are part of the most likely state sequence to *i*. Finally, we would like to incorporate an additional display that will provide a visualization of EM HMM training. We will use (Eisner, 2002) as a starting point.

References

Jason Eisner. 2002. An interactive spreadsheet for teaching the forward-backward algorithm. In *Proc. of the ACL 2002 Workshop on effective tools and methodologies for teaching natural language processing and computational linguistics*.

Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing: an introduction to natural language processing, and computational linguistics, and speech recognition*. Prentice-Hall.

J. Kupiec. 1992. Robust part-of-speech tagging using a hidden markov model. *Computer Speech and Language*, 6:225–242.

M. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, June.

Concrete Assignments for Teaching NLP in an M.S. Program

Reva Freedman
Department of Computer Science
Northern Illinois University
DeKalb, IL 60115
freedman@cs.niu.edu

Abstract

The professionally oriented computer science M.S. students at Northern Illinois University are intelligent, interested in new ideas, and have good programming skills and a good math background. However, they have no linguistics background, find traditional academic prose difficult and uninteresting, and have had no exposure to research. Given this population, the assignments I have found most successful in teaching Introduction to NLP involve concrete projects where students could see for themselves the phenomena discussed in class. This paper describes three of my most successful assignments: duplicating Kernighan et al.'s Bayesian approach to spelling correction, a study of Greenberg's universals in the student's native language, and a dialogue generation project. For each assignment I discuss what the students learned and why the assignment was successful.

1 Introduction

Northern Illinois University is a large public university (25,000 students) located in the farm-oriented exurbs of Chicago, about 60 miles west of the city. Most of the undergraduate computer science majors and about a third of the M.S. students come from the hi-tech corridor west of Chicago or small towns near the university. The remaining M.S. students are international students, currently mostly from India.

This paper discusses my experiences in two semesters of teaching Introduction to NLP and three semesters of teaching an NLP unit in an Introduction to Artificial Intelligence course. Because the students have no background in linguistics and are not used to reading the type of academic prose found in the textbook (Jurafsky and Martin, 2000), the most successful units I have taught involved concrete assignments where students could see for themselves the phenomena discussed in class. Successful assignments also did not assume any background in linguistics, even such basic notions as part of speech.

To provide an overview of the field, each year the NLP course contains three segments: one on a statistical approach to NLP, one on syntax, and one on a logic-based approach. The three segments are also chosen to include topics from phonology and morphology, syntax, and pragmatics. The specific content changes from year to year in an effort to find topics that both represent current issues in the field and capture the students' imagination.

This paper describes three of the most successful assignments. For each one, I describe the assignment, the topics the students learned, and why it was successful. The three assignments are: duplicating Kernighan et al.'s Bayesian approach to spelling correction, a study of Greenberg's universals in a language other than English (usually the student's native language), and a dialogue generation project using my research software.

2 Background

2.1 Student demographics

Most of the students taking Introduction to NLP are graduate students, although undergraduates are eligible if they have had three semesters of C++ programming. Graduate students in cognitive science-related fields, such as psychology or linguistics, are eligible if they have taken one semester of programming and are willing to teach themselves about trees. I actively recruit non-computer science students because it makes the course more interesting. In addition to providing a broader spectrum of interests, they tend to be more outgoing. They tend to be more willing to answer questions in class, and also to ask questions in class, which many of the computer science students will not do.

The preferred career path among the students is to obtain a programming job in local industry, preferably in a hi-tech area. However, among both undergraduates and graduate students, a few continue their education. One minority student with no previous experience in research became interested and is now planning to apply to a PhD program. In general, students take the course out of a desire to do something different from the normal operating systems, networking and database courses. An occasional student also takes the course because it fits in their schedule or because it doesn't have onerous prerequisites.

In general, the international students have good to near-native competence in spoken English, although a few cannot not follow my lectures, and some do not have sufficient writing skills for an essay exam. All could read my lecture notes without difficulty. Both among the international students and local students, many do not have sufficient experience with formal academic prose to understand the textbook (Jurafsky and Martin, 2000). Students' first languages have included Telugu, Hindi/Urdu, Nepali, Chinese (Mandarin), and Bulgarian.

2.2 Student background

Koedinger (2001), in his research on tutoring systems for high school mathematics, gives the following as his fundamental principle: "the student is not like me." In particular, student background frequently did not include the following items:

1) Parts of speech
2) Basic English grammar
3) Relationships between languages and language families
4) Practical issues, such as the importance of transliteration and glossing
5) Philosophical issues, such as the fact that there is no single authoritative grammar of a natural language or that one language is not more difficult than another in an absolute sense

However, students were talented at and enjoyed programming. Most students also had a good math background. Finally, they were enthusiastic about learning new things, as long as it involved concrete examples that they could work out and a sample problem with a solution that they could use as a model.

3 Spelling correction

3.1 Background

The goal of the first section of the course was to show the students the power of statistical methods in NLP. In this section, students were asked to duplicate the calculations used in Kernighan et al.'s (1990) Bayesian approach to spelling correction, as explained in Section 5.5 of the textbook.

Kernighan et al. choose as the preferred correction the one that maximizes $P(t|c)P(c)$, where t is the typo and c is a candidate correction. Candidate corrections are generated by assuming that errors involve only one letter or the transposition of two adjacent letters. To reproduce this calculation, students need the confusion matrices provided in the original paper, a source of unigram and bigram data, and a source for word frequencies.

3.2 Assignment

Students are given some misspelled words and possible corrections, such as the following examples from Kernighan et al:

misspelled word	possible corrections
ambitios	ambitious
	ambitions
	ambition

For each of these misspelled words, students are asked to do the following:

a) Use the method described by Kernighan et al., or equivalently in section 5.5 of the text, to find the probability of each possible correction.

b) Use their preferred spell checker (Microsoft Word, Unix ispell, etc.) to generate possible corrections for the same misspelled words.

The following questions are asked for each misspelled word:

• Is the most probable correction according to Kernighan's algorithm the same as the one suggested by your program?

• Which additional possible corrections (i.e., non-single-error corrections or non-single word corrections) does your program generate?

• Which of Kernighan's possible corrections does your program omit?

Since Kernighan's original paper omits the unigram and bigram count matrices, I provide a file with this information. Students are encouraged to find a source for word frequencies on the Web. As one option, I suggest they use any search engine (e.g., Google), after class discussion about the approximations involved in this approach.

Students are also given two summary questions to answer:

• A former student, Mr. I. M. Kluless, says: I don't see the point of using the frequency of potential corrections in the corpus (i.e., the prior probability) as part of Kernighan's algorithm. I would just use the likelihood of a given error. How would you answer Mr. Kluless? (One way to think about this question is: what would happen if you left it out?)

• Another former student, Ms. U. R. Useless, says: I don't see the point of using the likelihood of a given error as part of Kernighan's algorithm. I would just use the prior probability. How would you answer Ms. Useless?

3.3 Results

Students enjoyed this assignment because it was straightforward and used mathematics they were familiar with. They were uniformly surprised to discover that spelling correction is generally done today using Bayesian concepts rather than by dictionary lookup alone. They were also surprised to learn that learn that results were largely independent of the corpus chosen. Students who already knew Bayes' theorem learned about an application completely different from the ones they had used in other courses.

The majority of students used my suggestion to approximate word frequencies in a corpus by page counts in Google. They were surprised to learn that in spite of the number of ways in which the web differs from an ideal corpus, the volume of data available ensures that accurate results are still obtained. The better students searched the web for corpora they preferred, including the works of Shakespeare and an online interface to the British National Corpus (http://sara.natcorp.ox.ac.uk/lookup.html).

4 Syntax and language universals

4.1 Background

The second section of the course had as its goal to teach the students some basic aspects of syntax. I started with parts of speech and basic concepts of context-free grammars. I then introduced unification grammars as a way of obtaining more power with fewer rules.

As a way of showing the syntactic variation among languages, I also introduced some of Greenberg's word order universals (Greenberg, 1966), following the exposition in Baker (2001). Although identifying the most probable underlying word order (SVO, etc.) of an unknown language can involve significant linguistic intuition, I did not expect students to achieve that goal. Rather, I used Greenberg's ideas to make students think about the rules they were generating instead of generating S --> NP VP by rote. Additionally, the use of multiple languages contributed to the university's goal of introducing ideas of internationalization and diversity in classes where feasible.

4.2 Assignment

The students were asked to prepare a 15-minute class presentation showing two or three interesting phenomena of one of the languages of the world. Most students used their native language.

They were asked to include the following information:

- Where the language fits in Greenberg's classification (SVO, etc.)
- One or more syntactic phenomena that make the language interesting
- A grammar fragment (a set of CFG rules, possibly with unification-based features) illustrating one of the chosen phenomena

They could show several interesting phenomena with a short implementation of one, a complex phenomenon and a longer fragment of grammar, or one interesting phenomenon and multiple ways to implement it.

For each example they used, they were required to show the original transliterated into the Roman alphabet, a morpheme-level analysis, and a translation into English.

As a template, I gave a presentation using a language none of them had been exposed to, modern Hebrew. The four sample phenomena I presented were: a) there is no indefinite article, b) nouns and adjectives must agree in gender and number, c) adjectives follow the noun, and d) the definite article is attached to every adjective in an NP as well as to the noun.

In addition to providing an example of the scope required, the presentation also introduced the students to conventions of linguistic presentation, including interlinear display of transliteration, morpheme analysis, and translation. One slide from my presentation is shown below:

```
he- khatul   ha- gadol
DET cat-M-S  DET big-M-S
“the big cat”
```

```
he- khatulim   ha- g’dolim
DET cat-M-PL   DET big-M-PL
“the big cats”
```

4.3 Results

This assignment was useful for ensuring that students had a basic grasp of many elements of syntax covered in Section II of the textbook, including parts of speech, context-free grammars, and unification grammars. Second, the class presentations provided students concrete examples of some major syntactic concepts that all languages share, as well as some of the differences. Finally, this assignment enabled students to learn about and present some of the core linguistic features of their native language.

5 Dialogue generation

5.1 Background

The third segment of the course had as its goal to show how a logic-based approach is useful in NLP. Since some of my previous work involves implementing dialogue software using a logic-based approach, dialogue systems was a natural choice for this segment.

Phenomena discussed in lecture included the concepts of speech act and discourse intention, the relationship between syntactic form and intention, direct and indirect speech acts, and a short introduction to dialogue act classification.

As a counterbalance to the more theoretical material from Greenberg, this section included some information about current commercial uses of NLP. Students were asked to read an article from the popular press (Mount, 2005) describing experiences with currently available commercial systems.

I used my own software, APE (Freedman, 2000), a domain-independent dialogue plan interpreter based on reactive planning concepts. APE uses a rule-based macro language implemented in Common Lisp. It is a hierarchical task network (HTN) style planner, achieving each goal via a series of subgoals. APE's high-level planning loop alternates between waiting for user input and planning responses. It executes plan operators until a primitive, non-decomposable one is obtained. In addition to *elicit* and *inform*, plans can also include primitives to query and update APE's internal knowledge base, thus giving the system a "mind." Primitives are added to a buffer until a primitive requiring a response from the user is received. At that point the operators in the buffer are used to build the output text. Goals are represented using first-order logic without quantifiers, with full unification used for matching.

APE provides two ways to change a plan in progress. The author can instruct the system either to switch from one method of satisfying a goal to another or to add new goals at the top of the agenda, possibly replacing existing goals. The latter facility is particularly useful in dialogue generation, since it allows the system to prompt the user after an error. This feature makes APE more powerful than the pushdown automaton one might use to implement a context-free grammar. In addition, APE is obviously more powerful than the finite-state machines often used in dialogue generation.

Use of APE allows students to generate realistic hierarchically structured conversations with a reasonable number of rules.

5.2 Assignment

Sample code presented in class involved looking up data in a database of presidents' names. The sample system prompted the user for input, then provided answers, error messages, and re-prompts as appropriate. As an illustration of the power of the approach, I also demonstrated some of my research software, which showed conversations embedded in a variety of front-end GUIs.

For the assignment, students were asked to choose their own topic. They were asked to choose a problem, then provide a database layout and draw a graph showing the possible conversations their system could generate. Finally, they were asked to implement the code. At the end of the semester, students made a five-minute presentation to the class showing their application.

5.3 Results

Students greatly enjoyed this assignment because it involved the activity they enjoyed most, namely programming. Even though it was qualitatively different from other algorithms they had learned, they had no trouble learning the unification algorithm, both iterative and recursive versions, because they were experienced in learning algorithms. For most students in our program, this project will be their only experience with a non-imperative programming language.

Students were not bothered by the fact that the sample software provided included some features not discussed in class. In fact, some of the better students studied these features and used them in their own programs.

Every student mastered the basics of logic programming, including how to choose between alternatives, establish a default, implement multi-step and hierarchical procedures, interact with the user, and access an external database. They also learned how to use unification along with multiple first-order relations to access and update a database. The weaker students simply used the sample software as a guide, while the stronger ones mastered the underlying concepts and wrote more creative code.

Student projects ranged the gamut, including a system for running a car dealership, a game identifying movie directors, and an interactive system providing health information.

6 Conclusions

Teaching NLP to students for whom this will be the only exposure to the topic, and possibly the only exposure to a research-oriented topic, can be a successful and enjoyable experience for both students and teacher. With good organization, students can do useful projects even in one semester.

One factor that has increased student satisfaction as well as their mastery of the material is the use of concrete assignments where students can see for themselves concepts described in class. Three such assignments I have successfully used involve duplicating Kernighan et al.'s Bayesian approach to spelling correction, a study of Greenberg's universals in the student's native language, and a dialogue generation project using my research software. Each of these assignments is used in one of the three segments of the course: statistical approaches to language, introduction to syntax, and logic-based approaches to NLP.

Acknowledgments

Michael Glass of Valparaiso University graciously supplied the unigram and bigram counts needed to implement Kernighan et al.'s (1990) spelling correction algorithm.

References

Baker, M. (2001). *Atoms of Language: The Mind's Hidden Rules of Grammar.* New York: Basic Books.

Freedman, R. (2000). Using a Reactive Planner as the Basis for a Dialogue Agent. In Proceedings of the Thirteenth Florida Artificial Intelligence Research Symposium (FLAIRS 2000), Orlando.

Greenberg, J. (1966). Some universals of grammar with particular reference to the order of meaningful elements. In *Universals of language*, ed. J. Greenberg, pp. 73–113. Cambridge, MA: MIT Press. 2nd ed.

Kernighan, M., Church, K., and Gale, W. (1990). A spelling correction program based on a noisy channel model. In COLING '90 (Helsinki), v. 2, pp. 205–211. Available online from the ACL archive at http://acl.ldc.upenn.edu/C/C90/C90-2036.pdf.

Koedinger, K. (2001). The Student is Not Like Me. In Tenth International Conference on Artificial Intelligence in Education (AI-ED 2001). San Antonio, TX. Keynote address. Slides available online at http://www.itsconference.org/content/seminars.htm.

Mount, I. (2005). Cranky Consumer: Testing Online Service Reps. *Wall Street Journal*, Feb. 1, 2005.

Language Technology from a European Perspective

Hans Uszkoreit, Valia Kordoni
Dept. of Computational Linguistics
Saarland University
D-66041, Saarbruecken, Germany
{uszkoreit, kordoni}@coli.uni-sb.de

Vladislav Kubon
UFAL MFF UK
Charles University
Prague, Czech Republic
vk@ufal.mff.cuni.cz

Michael Rosner
Dept. of Computer Science and A.I.
University of Malta
Msida, Malta
mike.rosner@um.edu.mt

Sabine Kirchmeyer-Andersen
Dept. of Computational Linguistics
Copenhagen Business School
Copenhagen, Denmark
ska.id@cbs.dk

Abstract

This paper describes the cooperation of four European Universities aiming at attracting more students to European master studies in Language and Communication Technologies. The cooperation has been formally approved within the framework of the new European program "Erasmus Mundus" as a Specific Support Action in 2004. The consortium also aims at creating a sound basis for a joint master program in the field of language technology and computer science.

1 European higher education: Erasmus Mundus

The Erasmus Mundus programme [1] is a co-operation and mobility program in the field of higher education. It aims to enhance quality in European higher education and to promote intercultural understanding through co-operation with non-EU countries.

The program is intended to strengthen European co-operation and international links in higher education by supporting high-quality European Masters Courses, by enabling students and visiting scholars from around the world to engage in postgraduate study at European universities, as well as by encouraging the outgoing mobility of European students and scholars towards non-EU countries.

The Erasmus Mundus program comprises four concrete actions:

ACTION 1 - Erasmus Mundus Masters Courses: high-quality integrated courses at masters level offered by a consortium of at least three universities in at least three different European countries.

ACTION 2 - Erasmus Mundus scholarships: a scholarship scheme for non-EU-country graduate students and scholars from the whole world.

ACTION 3 - Partnerships: Erasmus Mundus Masters Courses selected under Action 1 also have the possibility of establishing partnerships with non-EU-country higher education institutions.

ACTION 4 - Enhancing attractiveness: projects aimed at enhancing the attractiveness of the European higher education.

2 LATER

One of the projects approved for funding (and the only one in the field of language technology) in the 2004 call is called LATER – **La**nguage **T**echnology **Er**asmus Mundus [2].

LATER falls under action 4 of the program and hence addresses the need to enhance the attractiveness of European higher education in Language

Technology and Communication (LCT). This need will be met through dissemination of the combined LCT-related expertise in of a consortium of Universities whose members are as follows

Saarland University in Saarbruecken (CoLi)

The Department of Computational Linguistics and Phonetics (CoLi) of Saarland University (coordinator) has an excellent international reputation for graduate training in Language Technologies, and for leading-edge basic research in this area. CoLi offers a new M.Sc. program in Language Science and Technology [3]. This is an active program of basic, applied and cognitive research, which combines with state-of-the-art facilities to provide students with a rich and stimulating environment for their research. Moreover, CoLi offers a European Ph.D. program in Language Technology and Cognitive Systems. In the past 15 years, CoLi has provided postgraduate research training to 100 early-stage researchers [4].

Charles University, Prague (ÚFAL)

The Institute of Formal and Applied Linguistics (ÚFAL) at the Faculty of Mathematics and Physics of the Charles University in Prague offers a five-year master program in Computer Science with several specialized branches. One of the branches of this program is the masters in Computational and Formal Linguistics [7]. It focuses mainly on the following four topics: formal description of natural language, grammars and automata in linguistics, methods of artificial intelligence in linguistics, as well as methods of automatic natural language processing.

University of Malta (UoM)

The Department of Computer Science and Artificial Intelligence at the University of Malta, established in 1993, teaches both Bachelors and Masters degree programs. The 4-year BSc. (Hons) scheme include several streams relevant to Language Technology including NLP and Computational Linguistics itself, Information Retrieval, Semantic Web, Internet and Agent technologies. The Department also runs a, one-year research oriented M.Sc. program [10]. The areas of specialization include the development of computational tools, techniques and resources for Maltese, the only semitic language to enjoy official EU status.

Copenhagen Business School (CBS)

The Department of Computational Linguistics is part of the Faculty of Modern Languages at the Copenhagen Business School. The Department is actively involved in research in the following four core fields: formal descriptions of the Danish language, modeling of knowledge relevant for LSP, LSP databases, and Machine Translation. Embedded in this context is the Master of Language Administration (MLA) [9] that the Department of Computational Linguistics of the Copenhagen Business School offers in co-operation with the University of Southern Denmark in Roskilde

3 Overall aims of the project

The overall aim of the project is to export the common educational experience currently embodied within existing Masters programs of the consortium to scholars and students of non-EU countries.

This aim will be realized by several different classes of activity under the rubrics of (i) workshops (ii) distance learning tools and (iii) coordination of a common Master program. We discuss these in the following sections.

3.1 Workshops

One of the most important types of activities of the project is organizing workshops and courses both for students from non-EU countries and for their teachers. The effect of these events is at least twofold – the students from countries or regions which do not have an access to any higher degree education in LCT get a chance to broaden their perspective by listening to lectures of prominent scientists and lecturers. The courses will also help the consortium to establish better contacts with non-EU Universities, teachers, and students which will turn out to be invaluable when disseminating the common European Master program in Language Technology discussed further below.

Both ÚFAL and CoLi have a long tradition in respect of offering such courses to students from the broadest possible range of countries.

ÚFAL has devoted a huge effort in the past to raise funding for the organization, once or twice a year, of a series of lectures by prominent scientists and lecturers from all over the world. This series of lectures, the Vilem Mathesius courses [6], have become well-known, especially among the Central and East European students of computational and general linguistics.

This year's course, held in March under the auspices of LATER, was able to support the attendance of 50 students from Russia, Ukraine, Albania, Bosnia, Serbia, Croatia and Georgia to lectures by prominent individuals including two ACL award winners.

At CoLi, the Computational Linguistics Colloquium is also a traditional event attracting the attention of both well-known lecturers and a number of master and postgraduate students from various countries. A second series of lectures in the frame of our project was held at the University of Saarlandes in Saarbruecken in January.

A third event, organized by the CBS, will take place in June. The first day consists of information seminar on content management and language technology to promote CBS' newly-launched International Master of Language Administration, whilst the second will be devoted to diffusion of a various issues connected to the Erasmus Mundus course.

Finally, a fourth event, in the form of a workshop with invited guest lecturers, is being organized at the University of Malta that will take place in September 2005. The theme of the workshop will be Machine Translation which is currently very topical given the newly-achieved official European status that the local language now enjoys.

3.2 Coordination of Masters Programs

A second important aim of the LATER project is the definition, coordination and implementation of an integrated European Masters Programme in LCT by creating a common basis that will appeal to both European and non-EU students.

The rationale behind the creation of such a programme is the assumption that LCT now occupies a central position in research and education in Europe, being a key enabling technology for numerous applications related to the information society, although the shortage of qualified researchers and developers is slowing down the speed of innovation in Europe.

The proposed programme addresses this shortage by creating a directed education and training opportunity for the next generation of LCT innovators in that will in turn bring educational, social and economic benefits. Some specific aims of Erasmus Mundus are also addressed: European education in LCT will be promoted worldwide and its competitiveness increased, increasing at the same time the competitiveness of European IT industries, creating a multilingual information society that is accessible for all, and turning the ``information overload" into a wealth of accessible and useful knowledge.

3.3 Distance learning tools

A third aim of LATER is the development of effective methods of hosting and integrating non-EU students, for example by developing distance learning tools and joint distance education modules, in order to facilitate outreach by online dissemination of courses. An example of such modules, as well as for computer-based tools, is being developed on the basis of the virtual courses CoLi has developed in the last 3 years in the framework of the MiLCA project (Medienintensive Lehrmodule in der Computerlinguistik-Ausbildung[1]).

We also plan to explore the use of collaboration technologies based on Sitescape [16], that have been developed at CBS for academic collaboration, for the management of certain aspects of the proposed Masters programme.

The fruits of various initiatives already under way at UoM will be exploited and extended during the life of the proposed course. These include interactive web based course delivery [13], just-in-time support based on P2P architectures [14], XML-based frameworks for online courses [15], the latter being developed within as a part of the Mediterranean Virtual University (MVU) EUMEDIS project [17].

4 Integrated European LCT Masters Programme

Whilst many agree with the above assessment of the importance of LCT, they disagree on the definition of "integrated course". Fortunately, we can turn to the comprehensive definition supplied by the EU call, the central element of which is "a jointly developed curriculum or full recognition by the consortium of modules which are developed

[1] for more see `http://milca.sfs.uni-tuebingen.de/index.html.`

and delivered separately, but make up a common standard Masters course."

Again, some turn away in horror at the notion of a standard curriculum in this area, the claim being that there is already enough standardization in the world, so why add to it? The point is, *any* programme dealing with LCT has to address the fact that it is highly interdisciplinary, including, at the core, computer science, computational and theoretical linguistics, and mathematics, and at the periphery, a wide variety of other subjects including electrical engineering, psychology, cognitive science artificial intelligence etc.

With such a large number of disciplines involved, it is practically impossible for a single University to excel in *all* of them. However if more than one University is involved, various kinds of curriculum sharing can be envisaged and so a much higher level of coverage becomes entirely achievable.

Put another way, curriculum sharing, together with common admission and assessment procedures envisaged, allows delivery of a complex course to be handled by what is effectively a "superuniversity".

4.1 Integration in practice

To put this idea into practice we are proposing that students will get the chance to attend a two years' master program at two universities chosen from a larger consortium, which is currently being put together. It includes the four original partners of the LATER project and the following new partners: University of Amsterdam (UvA) in the Netherlands, Free University of Bolzano-Bozen (FUB) in Italy, the Universities of Nancy 1 and Nancy 2 in France, Roskilde University in Denmark and Utrecht University in the Netherlands.

Studying in multi-national groups at two universities in Europe, with English as instruction language, accompanied by language classes in another European language, will contribute to the students' preparation for the increasing globalization of science, commerce and industry. The course also will also prepare students for follow-up Ph.D. studies provided by the participating partners and others.

The proposed programme follows the Bologna model for higher education in Europe and comprises 120 ECTS[2] credits, 30 of which make up the Masters dissertation, and 90 of which are coursework credits structured as follows:

- Compulsory modules in Computer Science (28 ECTS)
- Compulsory modules in Language Technology (28 ECTS)
- Advanced modules in Language Technology, Computational Linguistics and Computer Science (34 ECTS)

Coursework is distributed over three semesters, while the dissertation is supposed to be completed in the fourth semester

It is important to underline that this structure permits a considerable degree of variation. First, a module might be "implemented" by different set of courses at different Universities. Secondly, the advanced modules are electives, based on the specific strengths in research and teaching of individual partner institutions. There is no requirement that the advanced modules offered by different Universities should coincide.

Let us now introduce individual modules in more detail. Parentheses indicate ECTS credits.

Computer Science Modules

The Computer Science Modules are as follows:

- **Logic, Computability and Complexity (≥ 9)** Topics: Logic & inference; Computability theory; Complexity theory; Discrete mathematics
- **Formal Languages and Algorithms (≥ 9)** Topics: Formal grammars and languages hierarchy; Parsing and compiler design; Search techniques and constraint resolution; Automated Learning
- **Data Structures, Data Organization and Processing (≥ 6)** Topics: Algebraic data types; Relational databases; Semi-structured data and XML; Information retrieval; Digital libraries
- **Advanced Modules and Applications(≥ 6)** Topics: Artificial Intelligence, Knowledge Ŕepresentation, Automated Reasoning, Semantic Web, Neural Networks, Machine Learning etc. Students are expected to obtain at least 9 ECTS credits from each of the first two

[2] European Credit Transfer System: a standard measure that is used in Europe for comparing the size of courses.

modules and 6 ECTS credits from each of the remaining two modules.

Language Technology Modules

The Language Technology Modules are these:

- **Foundations of Language Technology (≥ 6)** Topics: Statistical methods; Symbolic methods; Cognition; Corpus Linguistics; Text and speech; Foundations of Linguistics
- **Computational Syntax and Morphology (≥ 9)** Topics: Finite state methods; Probabilistic approaches; Formal grammars; Tagging; Chunking; Parsing
- **Computational Semantics, Pragmatics and Discourse (≥ 6)** Topics: Syntax-semantics interface; Semantic construction; Dialogue; Formal semantics
- **Advanced Modules and Applications (≥ 6)** Topics: Machine Translation, Information Retrieval, Speech Recognition, Question Answering, Psycholinguistics etc..

4.2 Main issues to be addressed

Although it was not explicitly mentioned in the previous text, the integration of existing master programmes is done exclusively pair-wise. The students can't study at three universities (although the rules of the Erasmus Mundus programme allow such triangular cooperation). The restrictions within our consortia go even further – the students do not have a free choice of a combination of any two universities from within the consortium, they must choose one of the pairs offered by the consortium.

The reason for such a restriction is pretty simple - it turned out that although all members of the consortia in principle provide education both in Computer Science and in Computational Linguistics, they differ in the balance between these two fields. Within the consortium, there are universities with a strong stress on a Computer Science courses, aiming at a complex education including the sound theoretical background in the field, while other universities offer a more practically oriented educational scheme, stressing the concepts attracting a wider audience, e.g. various types of web technologies, databases, data mining etc.

As a result of this, each university participates in an average of four bilateral partnerships. We think that the fact that the consortium consists of universities which are not identical greatly increases the variety of options available. They have a chance to choose those universities which are best suited to their preferences whether these are in terms of subject area emphasis or geographical region.

The preparation of the integrated Master programme doesn't stop at matching the universities and lectures offered. Erasmus Mundus is not just a cooperation, it is really a completely new scheme which must also address practical issues as grades, examination procedures, admission procedure, tuition fees, defense of the thesis, local specialties existing at some partner universities etc.

The proposed Masters programme is something new. It is the first attempt to create a comprehensive Masters degree in this subject area that conforms to all the legalistic requirements of each participating University. Students completing the course will possess a Masters degree delivered by two of the participant Universities. This is in contrast to the existing European Master in Language and Speech [11], which is implemented through a certification procedure that does not replace any legal degree that a student may obtain from a University.

5 Conclusion

Although the process of establishing a new European Master programme in Language Technology was really very complicated, time consuming and painful, there are definitely already at this stage very positive results.

In order to submit a proposal, our consortium has managed to overcome all formal and structural differences among all partners, it has found a reasonable model of cooperation, it has developed a high-quality master programme open both to European and non-EU students.

The wide variety of modules and topics offered combined with a relatively high degree of freedom of choice for students allows for individual pairs of partner universities to promote those courses and fields in which they excel. The students are of course offered individual guidance from consortium members in order to allow them to identify that pair of universities which best suits their individual needs and preferences

The strategy we have chosen – the initial cooperation of a smaller consortium in the LATER project, promoting LTC education among the students from outside the EU and testing our ability both to offer a coordinated high-quality education and to attract a reasonable amount of interested students, has turned to be a sound one. It also helped to solve some issues in the larger consortium based on the experience from the smaller one.

References

[1] http://europa.eu.int/comm/education/programmes/mundus/index_en.html (Erasmus Mundus web page)

[2] http://europa.eu.int/comm/education/programmes/mundus/projects/2004/47.pdf (The description of the LATER project)

[3] http://www.coli.uni-saarland.de/msc/ (the MSc website at the University of Saarlandes in Saarbruecken)

[4] http://www.coli.uni-saarland.de/kvv/ (courses at the Dept. of Computational Linguistics at the University of Saarlandes in Saarbruecken)

[5] http://www.coli.uni-saarland.de/courses/late2/ (the web page of the Language Technology II course in Saarbruecken)

[6] http://ufal.mff.cuni.cz/vmc/vmc_ls20.html (the web page of the Vilem Mathesius Lecture Series)

[7] http://www.mff.cuni.cz/toUTF8.en/studium/bcmgr/ok/i1b53.htm (the master programme in Mathematical Linguistics at the Charles University in Prague)

[8] http://web.cbs.dk/stud_pro/clmdatauk.shtml (the master program at the Copenhagen Business School)

[9] http://uk.cbs.dk/mla (Master of Language Administration at the Copenhagen Business School)

[10] http://www.cs.um.edu.mt/research/pgEnquiries.html (the master program at the University of Malta)

[11] http://www.cstr.ed.ac.uk/euromasters (European Masters in Language and Speech)

[12] A.Burchardt, S. Walter and M. Pinkal. 2004. "MiLCA -- Distance Education in Computational Linguistics". In Szucs, Andras and Bo, Ingeborg (eds.), New Challenges and Partnerships in an Enlarged European Union – Proc. 2004 EDEN Conference, Budapest, pp. 351-356.

[13] Ellul, C., 2002, "Just-in-Time Lecture Delivery, Management and Student Support System", BSc. Project report, Dept. CSAI, University of Malta.

[14] Bezzina, R., 2002, "Peer-to-Peer Just-in-Time Support for Curriculum based Learning", BSc. Project report, Dept. CSAI, University of Malta.

[15] Cachia, E., and Micallef, M., forthcoming, "A Universal XML/XSLT Framework for Online Courses", Proc. International Conference on IT-Based Higher Education And Training (ITHET)", Dominican Republic.

[16] www.sitescape.com : SiteScape corporate website.

[17] http://www.eumedis.net/en/project/22: Mediterranean Virtual University (MVU) description.

Natural Language Processing at the School of Information Studies for Africa

Björn Gambäck
Userware Laboratory
Swedish Institute of Computer Science
Box 1263, SE–164 29 Kista, Sweden
gamback@sics.se

Gunnar Eriksson
Department of Linguistics
Stockholm University
SE–106 91 Stockholm, Sweden
gunnar@ling.su.se

Athanassia Fourla
Swedish Program for ICT in Developing Regions
Royal Institute of Technology/KTH
Forum 100, SE–164 40 Kista, Sweden
afourla@dsv.su.se

Abstract

The lack of persons trained in computational linguistic methods is a severe obstacle to making the Internet and computers accessible to people all over the world in their own languages. The paper discusses the experiences of designing and teaching an introductory course in Natural Language Processing to graduate computer science students at Addis Ababa University, Ethiopia, in order to initiate the education of computational linguists in the Horn of Africa region.

1 Introduction

The development of tools and methods for language processing has so far concentrated on a fairly small number of languages and mainly on the ones used in the industrial part of the world. However, there is a potentially even larger need for investigating the application of computational linguistic methods to the languages of the developing countries: the number of computer and Internet users of these countries is growing, while most people do not speak the European and East-Asian languages that the computational linguistic community has so far mainly concentrated on. Thus there is an obvious need to develop a wide range of applications in vernacular languages, such as translation systems, spelling and grammar checkers, speech synthesis and recognition, information retrieval and filtering, and so forth. But who will develop those systems? A prerequisite to the creation of NLP applications is the education and training of computer professionals skilled in localisation and development of language processing resources. To this end, the authors were invited to conduct a course in Natural Language Processing at the School of Information Studies for Africa, Addis Ababa University, Ethiopia. As far as we know, this was the first computational linguistics course given in Ethiopia and in the entire Horn of Africa region.

There are several obstacles to progress in language processing for new languages. Firstly, the particulars of a language itself might force new strategies to be developed. Secondly, the lack of already available language processing resources and tools creates a vicious circle: having resources makes producing resources easier, but not having resources makes the creation and testing of new ones more difficult and time-consuming.

Thirdly, there is often a disturbing lack of interest (and understanding) of the needs of people to be able to use their own language in computer applications — a lack of interest in the surrounding world, but also sometimes even in the countries where a language is used ("Aren't those languages going to be extinct in 50–100 years anyhow?" and "Our company language is English" are common comments).

And finally, we have the problem that the course described in this paper mainly tries to address, the lack of skilled professionals and researchers with knowledge both of language processing techniques and of the domestic language(s) in question.

The rest of the paper is laid out as follows: The next section discusses the language situation in Ethiopia and some of the challenges facing those trying to introduce NLP methods in the country. Section 3 gives the background of the students and the university, before Section 4 goes into the effects these factors had on the way the course was designed.

The sections thereafter describe the actual course content, with Section 5 being devoted to the lectures of the first half of the course, on general linguistics and word level processing; Section 6 is on the second set of lectures, on higher level processing and applications; while Section 7 is on the hands-on exercises we developed. The evaluation of the course and of the students' performance is the topic of Section 8, and Section 9 sums up the experiences and novelties of the course and the effects it has so far had on introducing NLP in Ethiopia.

2 Languages and NLP in Ethiopia

Ethiopia was the only African country that managed to avoid being colonised during the big European power struggles over the continent during the 19th century. While the languages of the former colonial powers dominate the higher educational system and government in many other countries, it would thus be reasonable to assume that Ethiopia would have been using a vernacular language for these purposes. However, this is not the case. After the removal of the Dergue junta, the Constitution of 1994 divided Ethiopia into nine fairly independent regions, each with its own "nationality language", but with Amharic being the language for countrywide communication. Until 1994, Amharic was also the principal language of literature and medium of instruction in primary and secondary schools, but higher education in Ethiopia has all the time been carried out in English (Bloor and Tamrat, 1996).

The reason for adopting English as the *Lingua Franca* of higher education is primarily the linguistic diversity of the country (and partially also an effect of the fact that British troops liberated Ethiopia from a brief Italian occupation during the Second World War). With some 70 million inhabitants, Ethiopia is the third most populous African country and harbours more than 80 different languages — exactly how many languages there are in a country is as much a political as a linguistic issue; the count of languages of Ethiopia and Eritrea together thus differs from 70 up to 420, depending on the source; with, for example, the Ethnologue (Gordon, 2005) listing 89 different ones.

Half-a-dozen languages have more than 1 million speakers in Ethiopia; three of these are dominant: the language with most speakers today is probably Oromo, a Cushitic language spoken in the south and central parts of the country and written using the Latin alphabet. However, Oromo has not reached the same political status as the two large Semitic languages Tigrinya and Amharic. Tigrinya is spoken in Northern Ethiopia and is the official language of neighbouring Eritrea; Amharic is spoken in most parts of the country, but predominantly in the Eastern, Western, and Central regions. Oromo and Amharic are probably two of the five largest languages on the continent; however, with the dramatic population size changes in many African countries in recent years, this is difficult to determine: Amharic is estimated to be the mother tongue of more than 17 million people, with at least an additional 5 million second language speakers.

As Semitic languages, Amharic and Tigrinya are distantly related to Arabic and Hebrew; the two languages themselves are probably about as close as are Spanish and Portuguese (Bloor, 1995). Speakers of Amharic and Tigrinya are mainly Orthodox Christians and the languages draw common roots to the ecclesiastic Ge'ez still used by the Coptic Church. Both languages use the Ge'ez (Ethiopic) script, written horizontally and left-to-right (in contrast to many other Semitic languages). Written Ge'ez can be traced back to at least the 4th century A.D. The first versions of the script included consonants only, while the characters in later versions represent consonant-vowel pairs. Modern Amharic words have consonantal roots with vowel variation expressing difference in interpretation.

Several computer fonts have been developed for the Ethiopic script, but for many years the languages had no standardised computer representation. An international standard for the script was agreed on only in year 1998 and later incorporated into Unicode, but nationally there are still about 30 different "standards" for the script, making localisation of language processing systems and digital resources difficult; and even though much digital information is now being produced in Ethiopia, no deep-rooted culture of information exchange and dissemination has been established. In addition to the digital divide, several other factors have contributed to this situation, including lack of library facilities and central resource sites, inadequate resources for digital production of journals and books, and poor documentation and archive collections. The difficulties of accessing information have led to low expectations and consequently under-utilisation of existing information resources (Furzey, 1996).

UNESCO (2001) classifies Ethiopia among the countries with "moribund or seriously endangered tongues". However, the dominating languages of the country are not under immediate threat, and serious efforts have been made in the last years to build and maintain linguistic resources in Amharic: a lot of work has been carried out mainly by Ethiopian Telecom, Ethiopian Science and Technology Commission and Addis Ababa University, as well as by Ethiopian students abroad, in particular in Germany, Sweden and the United States. Except for some initial efforts for the related Tigrinya, work on other Ethiopian languages has so far been scarce or nonexistent — see Alemu et al. (2003) or Eyassu and Gambäck (2005) for short overviews of the efforts that have been made to date to develop language processing tools for Amharic.

One of the reasons for fostering research in language processing in Ethiopia was that the expertise of a pool of researchers in the country would contribute to maintaining those Ethiopian languages that are in danger of extinction today. Starting with Amharic and developing a robust linguistic resource base in the country, together with including the Amharic language in modern language processing tools could create the critical mass of experience, which is necessary in order to expand to other vernacular languages, too.

Moreover, the development of those conditions that lay the foundations for language and speech processing research and development in the country would prevent potential brain drain from Ethiopia; instead of most language processing work being done by Ethiopian students abroad (at present), in the future it could be done by students, researchers and professionals inside the country itself.

3 Infrastructure and Student Body

Addis Ababa University (AAU) is Ethiopia's oldest, largest and most prestigious university. The Department of Information Science (formerly School of Information Studies for Africa) at the Faculty of Informatics conducts a two-year Master's Program. The students admitted to the program come from all over the country and have fairly diverse backgrounds. All have a four-year undergraduate degree, but not necessarily in any computer science-related subject. However, most of the students have been working with computers for some time after their under-graduate studies. Those admitted to the program are mostly among the top students of Ethiopia, but some places are reserved for public employees.

The initiative of organising a language processing course as part of the Master's Program came from the students themselves: several students expressed interest in writing theses on speech and language subjects, but the faculty acknowledged that there was a severe lack of staff qulified to teach the course. In fact, all of the university is under-staffed, while admittance to the different graduate programs has been growing at an enormous speed; by 400% only in the last two years. There was already an ICT support program in effect between AAU and SAREC, the Department for Research Cooperation at the Swedish International Development Cooperation Agency. This cooperation was used to establish contacts with Stockholm University and the Swedish Institute of Computer Science, that both had experience in developing computational linguistic courses.

Information Science is a modern department with contemporary technology. It has two computer labs with PCs having Internet access and lecture rooms with all necessary aids. A library supports the teaching work and is accessible both to students and staff. The only technical problems encountered arose from the frequent power failures in the country that created difficulties in teaching and/or loss of data. Internet access in the region is also often slow and unreliable. However, as a result of the SAREC ICT support program, AAU is equipped with both an internal network and with broadband connection to the outside world. The central computer facilities are protected from power failures by generators, but the individual departments have no such back-up.

4 Course Design

The main aim of the course plan was to introduce the students successfully to the main subjects of language and speech processing and trigger their interest in further investigation. Several factors were important when choosing the course materials and deciding on the content and order of the lectures and exercises, in particular the fact that the students did not have a solid background in either Computer Science or Linguistics, and the time limitations as the course could only last for ten weeks. As a result, a curriculum with a holistic view of NLP was built in the form of a "crash course" (with many lectures and labs per week, often having to use Saturdays too) aiming at giving as much knowledge as possible in a very short time.

The course was designed before the team travelled to Ethiopia, but was fine-tuned in the field based on the day-by-day experience and interaction with the students: even though the lecturers had some knowledge of the background and competence of the students, they obviously would have to be flexible and able to adjust the course set-up, paying attension both to the specific background knowledge of the students and to the students' particular interests and expectations on the course.

From the outset, it was clear that, for example, very high programming skills could not be taken for granted, as given that this is not in itself a requirement for being admitted to the Master's Program. On the other hand, it was also clear that *some* such knowledge could be expected, this course would be the last of the program, just before the students were to start working on their theses; and several laboratory exercises were developed to give the students hands-on NLP experience.

Coming to a department as external lecturers is also in general tricky and makes it more difficult to know what actual student skill level to expect. The lecturer team had quite extensive previous experiences of giving external courses this way (in Sweden and Finland) and thus knew that "the home department" often tends to over-estimate the knowledge of their students; another good reason for trying to be as flexible as possible in the course design. and for listening carefully to the feedback from the students during the course.

The need for flexibility was, however, somewhat counter-acted by the long geographical distance and time constraints. It was necessary to give the course in about two months time only, and with one of the lecturers present during the first half of the course and the other two during the second half, with some overlap in the middle. Thus the course was split into two main parts, the first concentrating on general linguistic issues, morphology and lexicology, and the second on syntax, semantics and application areas.

The choice of reading was influenced by the need not to assume very elaborated student programming skills. This ruled out books based mainly on programming exercises, such as Pereira and Shieber (1987) and Gazdar and Mellish (1989), and it was decided to use Jurafsky and Martin (2000) as the main text of the course. The extensive web page provided by those authors was also a factor, since it could not be assumed that the students would have full-time access to the actual course book itself. The costs of buying a regular computer science book is normally too high for the average Ethiopian student.

To partially ease the financial burden on the students, we brought some copies of the book with us and made those available at the department library. We also tried to make sure that as much as possible of the course material was available on the web. In addition to the course book we used articles on specific lecture topics particularly material on Amharic, for which we also created a web page devoted to online Amharic resources and publications.

The following sections briefly describe the different parts of the course and the laboratory exercises. The course web page contains the complete course materials including the slides from the lectures and the resources and programs used for the exercises:

`www.sics.se/humle/ile/kurser/Addis`

5 Linguistics and word level processing

The aim of the first part of the course was to give the students a brief introduction to Linguistics and human languages, and to introduce common methods to access, manipulate, and analyse language data at the word and phrase levels. In total, this part consisted of seven lectures that were accompanied by three hands-on exercises in the computer laboratory.

5.1 Languages: particularities and structure

The first two lectures presented the concept of a human language. The lectures focused around five questions: What is language? What is the ecological situation of the world's languages and of the main languages of Ethiopia? What differences are there between languages? What makes spoken and written modalities of language different? How are human languages built up?

The second lecture concluded with a discussion of what information you would need to build a certain NLP application for a language such as Amharic.

5.2 Phonology and writing systems

Phonology and writing systems were addressed in a lecture focusing on the differences between writing systems. The SERA standard for transliterating Ethiopic script into Latin characters was presented. These problems were also discussed in a lab class.

5.3 Morphology

After a presentation of general morphological concepts, the students were given an introduction to the morphology of Amharic. As a means of handling morphology, regular languages/expressions and finite-state methods were presented and their limitations when processing non-agglutinative morphology were discussed. The corresponding lab exercise aimed at describing Amharic noun morphology using regular expressions.

In all, the areas of phonology and morphology were allotted two lectures and about five lab classes.

5.4 Words, phrases and POS-tagging

Under this heading the students were acquainted with word level phenomena during two lectures. Tokenisation problems were discussed and the concept of dependency relations introduced. This led on to the introduction of the phrase-level and N-gram models of syntax. As examples of applications using this kind of knowledge, different types of part-of-speech taggers using local syntactic information were discussed. The corresponding lab exercise, spanning four lab classes, aimed at building N-gram models for use in such a system.

The last lecture of the first part of the course addressed lexical semantics with a quick glance at word sense ambiguation and information retrieval.

6 Applications and higher level processing

The second part of the course started with an overview lecture on natural language processing systems and finished off by a final feedback lecture, in which the course and the exam were summarised and students could give overall feedback on the total course contents and requirements.

The overview lecture addressed the topic of what makes up present-day language processing systems, using the metaphor of Douglas Adams' Babel fish (Adams, 1979): "What components do we need to build a language processing system performing the tasks of the Babel fish?" — to translate unrestricted speech in one language to another language — with Gambäck (1999) as additional reading material.

In all, the second course part consisted of nine regular lectures, two laboratory exercises, and the final evaluation lecture.

6.1 Machine Translation

The first main application area introduced was Machine Translation (MT). The instruction consisted of two 3-hour lectures during which the following subjects were presented: definitions and history of machine translation; different types of MT systems; paradigms of functional MT systems and translation memories today; problems, terminology, dictionaries for MT; other kinds of translation aids; a brief overview of the MT market; MT users, evaluation, and application of MT systems in real life. Parts of Arnold et al. (1994) complemented the course book.

There was no obligatory assignment in this part of the course, but the students were able to try out and experiment with online machine translation systems. Since there is no MT system for Amharic, they used their knowledge of other languages (German, French, English, Italian, etc.) to experience the use of automatic translation tools.

6.2 Syntax and parsing

Three lectures and one laboratory exercise were devoted to parsing and the representation of syntax, and to some present-day syntactic theories. After introducing basic context-free grammars, Dependency Grammar was taken as an example of a theory underlying many current shallow processing systems. Definite Clause Grammar, feature structures, the concept of unification, and subcategorisation were discussed when moving on to more deeper-level, unification-based grammars.

In order to give the students an understanding of the parsing problem, both processing of artificial and natural languages was discussed, as well as human language processing, in the view of Kimball (1973). Several types of parsers were introduced, with increasing complexity: top-down and bottom-up parsing; parsing with well-formed substring tables and charts; head-first parsing and LR parsing.

6.3 Semantics and discourse

Computational semantics and pragmatics were covered in two lectures. The first lecture introduced the basic tools used in current approaches to semantic processing, such as lexicalisation, compositionality and syntax-driven semantic analysis, together with different ways of representing meaning: first-order logic, model-based and lambda-based semantics. Important sources of semantic ambiguity (quantifiers, for example) were discussed together with the solutions allowed by using underspecified semantic representations.

The second lecture continued the semantic representation thread by moving on to how a complete discourse may be displayed in a DRS, a Discourse Representation Structure, and how this may be used to solve problems like reference resolution. Dialogue and user modelling were introduced, covering several current conversational systems, with Zue and Glass (2000) and Wilks and Catizone (2000) as extra reading material.

6.4 Speech technology

The final lecture before the exam was the only one devoted to speech technology and spoken language translation systems. Some problems in current spoken dialogue systems were discussed, while text-to-speech synthesis and multimodal synthesis were just briefly touched upon. The bulk of the lecture concerned automatic speech recognition: the parts and architectures of state-of-the-art speech recognition systems, Bayes' rule, acoustic modeling, language modeling, and search strategies, such as Viterbi and A-star were introduced, as well as attempts to build recognition systems based on hybrids between Hidden Markov Models and Artificial Neural Networks.

7 Laboratory Exercises

Even though we knew before the course that the students' actual programming skills were not extensive, we firmly believe that the best way to learn Computational Linguistics is by hands-on experience. Thus a substantial part of the course was devoted to a set of laboratory exercises, which made up almost half of the overall grade on the course.

Each exercise was designed so that there was an (almost obligatory) short introductory lecture on the topic and the requirements of the exercise, followed by several opportunities for the students to work on the exercise in the computer lab under supervision from the lecturer. To pass, the students both had to show a working system solving the set problem and hand in a written solution/explanation. Students were allowed to work together on solving the problem, while the textual part had to be handed in by each student individually, for grading purposes.

7.1 Labs 1–3: Word level processing

The laboratory exercises during the first half of the course were intended to give the students hands-on experience of simple language processing using standard UNIX tools and simple Perl scripts. The platform was cygwin,[1] a freeware UNIX-like environment for Windows. The first two labs focused on regular expressions and the exercises included searching using 'grep', simple text preprocessing using 'sed', and building a (rather simplistic) model of Amharic noun morphology using regular expressions in (template) Perl scripts. The third lab exercise was devoted to the construction of probabilistic N-gram data from text corpora. Again standard UNIX tools were used.

Due to the students' lack of experience with this type of computer processing, more time than expected was spent on acquainting them with the UNIX environment during the first lab excercises.

7.2 Labs 4–5: Higher level processing

The practical exercises during the second half of the course consisted of a demo and trial of on-line machine translation systems, and two obligatory assignments, on grammars and parsing and on semantics and discourse, respectively. Both these exercises

[1] `www.cygwin.com`

consisted of two parts and were carried out in the (freeware) SWI-Prolog framework.[2]

In the first part of the fourth lab exercise, the students were to familiarise themselves with basic grammars by trying out and testing parsing with a small context-free grammar. The assignments then consisted in extending this grammar both to add coverage and to restrict it (to stop "leakage"). The second part of the lab was related to parsing. The students received parsers encoding several different strategies: top-down, bottom-up, well-formed substring tables, head parsing, and link parsing (a link parser improves a bottom-up parser in a similar way as a WFST parser improves a top-down parser, by saving partial parses). The assignments included creating a test corpus for the parsers, running the parsers on the corpus, and trying to determine which of the parsers gave the best performance (and why).

The assignments of the fifth lab were on lambda-based semantics and the problems arising in a grammar when considering left-recursion and ambiguity. The lab also had a pure demo part where the students tried out Johan Bos' "Discourse Oriented Representation and Inference System", DORIS.[3]

8 Course Evaluation and Grading

The students were encouraged from the beginning to interact with the lecturers and to give feedback on teaching and evaluation issues. With the aim of coming up with the best possible assessment strategy — in line with suggestions in work reviewed by Elwood and Klenowski (2002), three meetings with the students took place at the beginning, the middle, and end of the course. In these meetings, students and lecturers together discussed the assessment criteria, the form of the exam, the percentage of the grade that each part of the exam would bear, and some examples of possible questions.

This effort to better reflect the objectives of the course resulted in the following form of evaluation: the five exercises of the previous section were given, with the first one carrying 5% of the total course grade, the other four 10% each, and an additional written exam (consisting of thirteen questions from the whole curriculum taught) 55%.

While correcting the exams, the lecturers tried to bear in mind that this was the first acquaintance of the students with NLP. Given the restrictions on the course, the results were quite positive, as none of the students taking the exam failed the course. After the marking of the exams an assessment meeting with all the students and the lecturers was held, during which each question of the exam was explained together with the right answer. The evaluation of the group did not present particular problems. For grading, the American system was used according to the standards of Addis Ababa University (i.e., with the grades 'A+', 'A', ..., 'F').

9 Results

Except for the contents of the course, the main innovation for the Information Science students was that the bulk of the course reading list and relevant materials were available online. The students were able to access the materials according to their own needs — in terms of time schedule — and download and print it without having to go to the library to copy books and papers.

Another feature of the on-line availability was that after the end of the course and as the teaching team left the country, the supervision of the students' theses was carried out exclusively through the Internet by e-mail and chat. The final papers with the signatures of the supervisors were even sent electronically to the department. The main difficulty that had to be overcome concerned the actual writing of the theses; the students were not very experienced in producing academic text and required some distance training, through comments and suggestions, on the subject.

The main results of the course were that, based strictly on the course aims, students were successfully familiarised with the notion of NLP. This also led to eight students choosing to write their Master theses on speech and language issues: two on speech technology, on text-to-speech synthesis for Tigrinya and on speech recognition for Amharic; three on Amharic information access, on information filtering, on information retrieval and text categorisation, and on automatic text summarisation; one on customisation of a prototype English-to-Amharic transfer-based machine translation system; one on predictive SMS (Short Message Service) text

[2] `www.swi-prolog.org`

[3] `www.cogsci.ed.ac.uk/~jbos/doris`

input for Amharic; and one on Amharic part-of-speech tagging. Most of these were supervised from Stockholm by the NLP course teaching team, with support from the teaching staff in Addis Ababa.

As a short-term effect, several scientific papers were generated by the Master theses efforts. As a more lasting effect, a previously fairly unknown field was not only tapped, but also triggered the students' interest for further research. Another important result was the strengthening of the connections between Ethiopian and Swedish academia, with ongoing collaboration and supervision, also of students from later batches. Still, the most important long-term effect may have been indirect: triggered by the success of the course, the Addis Ababa Faculty of Informatics in the spring of 2005 decided to establish a professorship in Natural Language Processing.

10 Acknowledgments

Thanks to the staff and students at the Department of Information Science, Addis Ababa University, in particular Gashaw Kebede, Kinfe Tadesse, Saba Amsalu, and Mesfin Getachew; and to Lars Asker and Atelach Alemu at Stockholm University.

This NLP course was funded by the Faculty of Informatics at Addis Ababa University and the ICT support program of SAREC, the Department for Research Cooperation at Sida, the Swedish International Development Cooperation Agency.

References

Douglas Adams. 1979. *The Hitch-Hiker's Guide to the Galaxy*. Pan Books, London, England.

Atelach Alemu, Lars Asker, and Mesfin Getachew. 2003. Natural language processing for Amharic: Overview and suggestions for a way forward. In *Proceedings of the 10th Conference on Traitement Automatique des Langues Naturelles*, volume 2, pages 173–182, Batz-sur-Mer, France, June.

Douglas Arnold, Lorna Balkan, Siety Meijer, R. Lee Humphreys, and Louisa Sadler. 1994. *Machine Translation: An Introductory Guide*. Blackwells-NCC, London, England.

Thomas Bloor and Wondwosen Tamrat. 1996. Issues in Ethiopian language policy and education. *Journal of Multilingual and Multicultural Development*, 17(5):321–337.

Thomas Bloor. 1995. The Ethiopic writing system: a profile. *Journal of the Simplified Spelling Society*, 19:30–36.

Jannette Elwood and Val Klenowski. 2002. Creating communities of shared practice: the challenges of assessment use in learning and teaching. *Assessment & Evaluation in Higher Education*, 27(3):243–256.

Samuel Eyassu and Björn Gambäck. 2005. Classifying Amharic news text using Self-Organizing Maps. In *ACL 2005 Workshop on Computational Approaches to Semitic Languages*, Ann Arbor, Michigan, June. ACL.

Jane Furzey. 1996. Enpowering socio-economic development in Africa utilizing information technology. A country study for the United Nations Economic Commission for Africa (UNECA), African Studies Center, University of Pennsylvania.

Björn Gambäck. 1999. Human language technology: The Babel fish. Technical Report T99-09, SICS, Stockholm, Sweden, November.

Gerald Gazdar and Chris Mellish. 1989. *Natural Language Processing in Prolog*. Addison-Wesley, Wokingham, England.

Raymond G. Gordon, Jr, editor. 2005. *Ethnologue: Languages of the World*. SIL International, Dallas, Texas, 15 edition.

Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, Upper Saddle River, New Jersey.

John Kimball. 1973. Seven principles of surface structure parsing in natural languages. *Cognition*, 2(1):15–47.

Fernando C. N. Pereira and Stuart M. Shieber. 1987. *Prolog and Natural Language Analysis*. Number 10 in Lecture Notes. CSLI, Stanford, California.

Yorick Wilks and Roberta Catizone. 2000. Human-computer conversation. In *Encyclopedia of Microcomputers*. Dekker, New York, New York.

Stephen Wurm, editor. 2001. *Atlas of the World's Languages in Danger of Disappearing*. The United Nations Educational, Scientific and Cultural Organization (UNESCO), Paris, France, 2 edition.

Victor Zue and James Glass. 2000. Conversational interfaces: Advances and challenges. *Proceedings of the IEEE*, 88(8):1166–1180.

Teaching Language Technology at the North-West University

Suléne Pilon
`sktsp@puk.ac.za`

Gerhard B van Huyssteen
`ntlgbvh@puk.ac.za`

Bertus van Rooy
`ntlajvr@puk.ac.za`

Research Focus Area: Languages and Literature in the South African Context North-West University,
Potchefstroom
2531
South Africa

Abstract

The *BA Language Technology* program was recently introduced at the North-West University and is, to date, the only of its kind in South Africa. This paper gives an overview of the program, which consists of computational linguistic subjects as well as subjects from languages, computer science, mathematics, and statistics. A brief discussion of the content of the program and specifically the computational linguistics subjects, illustrates that the *BA Language Technology* program is a vocationally directed, future oriented teaching program, preparing students for both future graduate studies and a career in language technology. By means of an example, it is then illustrated how students and researchers alike benefit from working side by side on research and development projects by using a problem-based, project-organized approach to curriculum design and teaching.

1 Introduction

A new undergraduate teaching program, *BA Language Technology,* was recently introduced at the Potchefstroom Campus of the North-West University (NWU). The introduction of this program was motivated by two factors:
(a) a need within the Faculty of Arts to develop teaching programs that are relevant, vocationally directed, and future-oriented; and
(b) a need in the South African higher education system for capacity building in the field of in language technology (PanSALB & DACST, 2000).

To date, the BA Language Technology program is the only one of its kind in South Africa. It has therefore remained imperative that the program equips students adequately to fill positions in the emerging South African language technology industry. At the same time, students should be able to continue with graduate studies, and therefore the program had to be designed in such a way that students receive an academic training that incorporates a solid theoretical component alongside the need to get enough practical experience. These two imperatives are reflected in the program structure, and also in the project-based learning approach that we adopted.

2 Program Structure

After wide consultation with international and local role players and experts, a program was designed that combines language subjects and natural sciences (mainly computer science, mathematics and statistics) with a core group of computational linguistic and language technology subjects. This section offers an overview of the *BA Language Technology* program. An example of a typical program will be given and the modules which form part of the program will be discussed briefly.

The program has a basic core of compulsory modules, but allows some room for students to take modules based on personal interest and ability. A student who excels in computer programming can choose to take additional modules from that field after completing the compulsory modules. Students may also choose to take more language or mathematics modules after completing their compulsory modules. There are also a number of general formative subjects that all students at the University must take, which are not being discussed here. The basic course structure is presented in Table 1.

YEAR 1	**YEAR 2**	**YEAR 3**	**YEAR 4**
First semester	**First semester**	**First semester**	**First semester**
Modules	**Modules**	**Modules**	**Modules**
Computer Science (programming)	**Language Technology: Introduction**	**Introduction to NLP**	**Language technology: Internship**
2 Languages	1 Language	1 CHOICE	
Statistics (introduction)	Computer Science (programming)	2 General formative modules	
Mathematics	1 CHOICE		
Applied Mathematics			
2 General formative modules			
Second semester	**Second semester**	**Second semester**	**Second semester**
Modules	**Modules**	**Modules**	**Modules**
Computer Science (programming)	**Language Technology: Linguistics for language technology students**	**Language Technology: Speech applications**	**Advanced NLP**
1 Language	1 CHOICE	**Language Technology: Text applications**	**Language Technology Project**
Statistics (Inferential)	2 General formative modules	1 CHOICE	
1 CHOICE			

Table 1: *BA Language Technology* compulsory modules with choices

The various general formative modules offered at the university include academic literacy, study skills, computer literacy and information skills, philosophy and academic and scientific writing courses. The elective modules from which the students can choose are mathematics, computer science and languages. The languages from which the students can choose are Afrikaans, English or Setswana (regular university courses) or introductory courses (foreign language level) in two South African languages, Setswana and isiZulu and two foreign European languages, German and French.

Students are encouraged to take at least one South African language. This is motivated in part by trends in the macro-political environment. In government policy documents, such as the final language policy presented to cabinet, language technology is principally regarded as a means to promote multilingualism and increase access of information in a country with eleven official languages. In the context of the program itself, it is expected that students acquire and/or improve their proficiency in the various languages; students are also expected to develop basic knowledge of the structure of the particular languages. This basic knowledge is then developed further in the module "Linguistics for language technology students" (second year, first semester). The module includes components of phonetics, morphology and syntax, to enable students to learn how to do detailed linguistic data analysis.

In the first semester of the second year, students are introduced to Language Technology. An overview of the field of study is given and it is indicated how the knowledge students gained in the modules they have completed, should be put to use within the field. The course also focuses on the relationship between a more practical language technology orientation and a more theoretical natural language processing (NLP) orientation, to enable students to see the broader picture and develop a sense for the coherence of the teaching program.

Language technologies are the subject of two modules in the second semester of the third year. They spend equal amounts of time on speech-based technologies and text-based technologies. The focus of these courses is specific language technology applications. At any given time, there are a number of ongoing projects at the university. Students are involved in these projects, learning to develop the specific applications, but also developing general skills for other types of applications, within the framework of project-based learning, as will be outlined later in this paper. Students are expected to participate in ongoing projects on various levels, ranging from annotating corpora to intricate programming – depending on their aptitude and preferences.

This is followed by a six-month internship in the first semester of the fourth year, at an approved company or higher education or research institution. Apart from extending their training in the development of language technology applications, the internship is intended to let students get a "real world" experience in the language technology industry before they have to make career decisions.

In their final semester, students have to complete a supervised project, which fits in with current research at the university. It is important that students should be positive about this project and therefore students are consulted when project topics are chosen. In this stage of the program, students have very little class in order to enable them to work on their projects on a full-time base, which provides for more practical experience.

Students are introduced to Natural Language Processing in the first semester of the third year. This course focuses mainly on statistical techniques for the analysis of the kinds of phonetic, morphological and syntactic data that were introduced in the second semester of the second year. The logic is that students must be able to analyze data manually as linguists first, in order to develop an appreciation for the capabilities, power and limitations of statistical NLP methods.

An advanced NLP course is offered in the second semester after students have completed their internship and while they are working on their own projects. This course is tailored to the individual interests and needs of the students. The specific NLP techniques relevant to their projects, as well as problems they encountered during their internships, serve as guiding principles for the selection of content. At the same time, we incorporate a selection of hot topics in NLP research and some techniques for dealing with semantic data.

As computational linguistics is a relatively new field of study in South Africa, students and lecturers/researchers have to learn together, even by making mistakes and taking 'wrong' sidetracks during the learning process. In order to facilitate these circumstances, a problem-oriented and project-organized approach, based on the educational system developed at the Aalborg University, Denmark, since 1974 (Kjersdam & Enemark, 1994), was taken in the design of the curricula of the Language Technology and NLP modules. This means that the content of some of the Language Technology and NLP subjects vary from year to year, depending on the current project(s) being conducted at the university. However, by working alongside each other on research and development projects, both students and lecturers engage in active learning, proving to yield excellent results in the acquiring of knowledge in the field. The next section describes how various research and development projects are integrated in the undergraduate and graduate teaching programs, in order to facilitate hands-on, outcome-based learning.

3 Teaching Approach: Problem-Based, Project-Organized Learning

Problem-Based Learning (PBL; also called problem-oriented education) can be defined as learning "based on working with unsolved, relevant and current problems from society/real life… By analyzing the problems in depth the students learn and use the disciplines and theories which are considered to be necessary to solve the problems posed, i.e. the problem defines the subjects and not the reverse" (Kjersdam & Enemark, 1994: 16; cf. Schwartz *et al*., 2001; Macdonald, 2002). This approach is successfully implemented world-wide in the teaching of specifically more applied sciences, such as, *inter alia*, medicine (Albanese & Mitchell, 1993; Barrows and Tamblyn, 1980; Moore *et al*., 1994), and engineering (De Graaf & Kolmos, 2003; Fink, 2002).

Within the context of computational linguistics, this "applied-teaching approach" maintains a dynamic triangular equilibrium between training, research and product development, serving researchers, students, and the industry alike. A project-organized approach offers lecturers an opportunity to align course material with their research projects, while students are enabled to gain "comprehensive knowledge of the development of theoretical and methodological tools" (Kjersdam and Enemark, 1994: 17). Therefore, after completion of their formal studies, students should be able to contribute to research and the development of original paradigms to solve new and complex problems in the future.

In the *BA Language Technology* program, PBL is incorporated with project-organized education in two ways. On the one hand various *project-based modules* are included in the curriculum. For instance in the third year of study, the modules "Language Technology: Speech Applications" and "Language Technology: Text Applications" are introduced, where students have to develop various small modules for both speech and text technological applications (e.g. a simple rule-based stemmer). In the final year of study, the largest part of the year is spent on independent project work, which is conducted either at the university, or while doing an internship elsewhere. These projects are on a much larger scale than the third year projects, with the possibility to build on the work of the previous year (e.g. to develop a more sophisticated stemmer, using more advanced NLP techniques).

On the other hand, some of the other modules (e.g. the "Natural Language Processing" modules) are more *project-driven*, since they are organized around existing research projects. Students are mostly drawn in on a so-called "design-oriented" level, i.e. where they have to deal with "know-how problems which can be solved by theories and knowledge they have acquired in their lectures" (Kjersdam & Enemark, 1994: 7). After the project and the problems related to the project are explained to students, they get involved by collecting data, identifying possible/different solutions, formulating rules and algorithms, analyzing data, evaluating different components, etc. In this way they get know-how and experience in theoretical, methodological, and implementation issues.

This can be illustrated by a recent example, where work on a spelling checker project was integrated in the curricula of various modules. In this project, involving the development of spelling checkers for five different South African languages, a variety of NLP techniques were implemented in the various spelling checkers, depending on the orthographical complexity of and resources available for a specific language. For instance, languages such as Tswana and Northern Sotho have a relatively simple orthographical structure (in the sense that it is more disjunctive), and a straightforward lexicon-based approach to spelling checking therefore suffice for these languages. In contrast, Afrikaans, Zulu and Xhosa are orthographically more complex languages, requiring a spelling checking approach based on morphological analysis or decomposition, which is of course more interesting from a computational linguistic perspective. For all of these languages, almost no resources were available at the start of the project, posing a huge but interesting challenge (e.g. could available technologies for other languages, such as a Porter stemmer, be adapted for these languages?).

From the onset of the spelling checker project, students were involved in all aspects of the project. Using Jurafsky & Martin (2000) as a point of departure, students were introduced to the basic problems of spelling checking, relating it to the current project and specifically to the challenges posed by spelling checking for Afrikaans (e.g. productive concatenative compound formation, derivational word formation, etc.). Students were thoroughly involved in all discussions of the aims of the project, potential problems and possible solutions, as well as the general system architecture (i.e. students were involved on the design-oriented level). Students were therefore introduced to basic concepts such as tokenization, stemming, and Levenshtein Distance (for purposes of generating suggestions), within a real-world context.

After the planning and design phase, each student got involved in solving different problems of the project, e.g. developing a stemmer (using finite-state techniques) and a compound analyzer (using machine-learning techniques), the automatic generation of a lexicon, evaluating spelling checkers (within the broader context of the evaluation of NLP applications), etc. Although each student worked separately on different problems, they were forced to extend their experience by helping each other with their different tasks, thereby expanding their general knowledge and experience. In this way, students also came to learn that different problems call for different approaches: to use finite-state techniques for hyphenation in Afrikaans is simply to labor-intensive, while machine learning offers highly efficient solutions to the problem. An introduction to machine learning was therefore also introduced in the curriculum.

The advantages of this approach proved to be many: not only did the project benefit from the sub-projects of each of the students, but students got the feeling that they were involved in "important" and relevant work. They got the opportunity to apply the theoretical knowledge they acquired in the classes in a practical, hands-on environment, to improve their understanding of the theories and concepts of the study field, and to solve real-world problems. Additionally, members of staff were enabled to harmonize their research and teaching responsibilities, optimizing the quality and quantity of their outputs. Moreover, existing students were motivated to continue with their studies in computational linguistics on MA level (where they are working on more advanced problems), while undergraduate student numbers increased (which can be ascribed to a greater awareness of language technology in the community, brought about partially by media coverage of the project, focusing on the promotion of multilingualism and language empowerment).

4 Conclusion

Since its very inception, the *BA Language Technology* program at the North-West University was designed as a vocationally directed, future-oriented teaching program. A curriculum with a core of computational linguistic subjects, strengthened by a strong foundation in languages, computer science, mathematics, and statistics, equips students both with enough practical experience to start working in the industry, and with enough theoretical knowledge to continue with postgraduate studies.

By taking a problem-based, project-organized approach to curriculum design, students and researchers alike benefit from working side by side on research and development projects (as illustrated by the incorporation of a spelling checker project in the curricula of various subjects). The same approach is followed in other subjects, such as "Language Technology: Speech Applications", where students are working in collaboration with their lecturers on various speech-based projects. As new research projects are initiated, the curricula of the various subjects are adapted accordingly. For example, in 2005 a new research project on syntactic parsing commenced – consequently, new students are confronted with other problems than their predecessors, while still learning, for example, about the differences between linguistic and statistical approaches to NLP. With the help of students in the program and others involved, the program is constantly evaluated and adjusted accordingly, thereby ensuring that it delivers well-educated and informed students, prepared for the challenges of a career in language technology.

References

Albanese MA & Mitchell S. 1993. Problem-based learning: A review of literature on its outcomes and implementation issues. *Academic Medicine* 68, 52-81.

Barrows HS & Tamblyn RM. 1980. *Problem-Based Learning: An Approach to Medical Education*. New York: Springer Publishing Company.

De Graaff, E & Kolmos, A. 2003. Characteristics of Problem-Based Learning. International *Journal of Engineering Education* 19(5).

Fink, FK. 2002. Problem-Based Learning in engineering education: a catalyst for regional industrial development. *World Transactions on Engineering and Technology Education* 1(1): 29-32.

Jurafsky, D & Martin, JH. 2000. *Speech and language processing : an introduction to natural language processing*. Upper Saddle River: Prentice Hall, 2000.

Kjersdam F & Enemark S. 1994. *The Aalborg Experiment: Project Innovation in University Education*. Aalborg: Aalborg University Press.

Macdonald R. 2002. Problem-based learning: some references. Available at: [WWW:]www.ics.ltsn.ac.uk/pub/pbl [Accessed 5 May 2003].

Moore GT, Block SD, Briggs Style C & Mitchell R. 1994. The influence of the New Pathway curriculum on Harvard medical students. *Academic Medicine* 69, 983-989.

Pan South African Language Board (PanSALB) & Department of Arts, Culture, Science and Technology (DACST). 2000. The development of Human Language Technologies in South Africa: Strategic Planning. (Report by the joint steering committee). Pretoria: Government Printers. Available at: www.dac.gov.za/about_us/cd_nat_language/language_planning/hlt_strategic_plan/hlt_strategic_plan2.htm#policy [Accessed April 1, 2005].

Schwartz P, Mennin S & Webb G. 2001. *Problem-based Learning: Case Studies, Experience and Practice*. London: Kogan Page.

Hands-On NLP for an Interdisciplinary Audience

Elizabeth D. Liddy and Nancy J. McCracken
Center for Natural Language Processing
School of Information Studies
Syracuse University
liddy@syr.edu, njm@ecs.syr.edu

Abstract

The need for a single NLP offering for a diverse mix of graduate students (including computer scientists, information scientists, and linguists) has motivated us to develop a course that provides students with a breadth of understanding of the scope of real world applications, as well as depth of knowledge of the computational techniques on which to build in later experiences. We describe the three hands-on tasks for the course that have proven successful, namely: 1) in-class group simulations of computational processes; 2) team posters and public presentations on state-of-the-art commercial NLP applications, and; 3) team projects implementing various levels of human language processing using open-source software on large textual collections. Methods of evaluation and indicators of success are also described.

1 Introduction

This paper presents both an overview and some of the details regarding audience, assignments, technology, and projects in an interdisciplinary course on Natural Language Processing that has evolved over time and been successful along multiple dimensions – both from the students' and the faculty's perspective in terms of accomplishments and enjoyment. This success has required us to meet the challenges of enabling students from a range of disciplines and diverse experience to each gain a real understanding of what is entailed in Natural Language Processing.

2 A Course Within Multiple Curricula

The course is entitled Natural Language Processing and is taught at the 600 graduate course level in a School of Information Studies in a mid to large-size private university. While NLP is not core to any of the three graduate degree programs in the Information School, it is considered an important area within the Information School for both professional careers and advanced research, as well as in the Computer Science and Linguistic Programs on campus. The course has been taught every 1½ to 2 years for the last 18 years. While some aspects of the course have changed dramatically, particularly in regards to the nature of the student team projects, the basic structure – the six levels of language processing – has remained essentially the same, with updates to topics within these levels reflecting recent research findings and new applications.

3 Audience

At the moment, this is the only course offering on NLP within the university, but a second-level, seminar course, entitled Content Analysis Research Using Natural Language Processing, geared towards PhD students doing social science research on large textual data sets, will be offered for the first time in Fall 2005. Given that the current NLP course is the only one taught, it cannot, by necessity, have the depth that could be achieved in curricula where there are multiple courses. In a more extensive curriculum, courses provide a greater depth than is possible in our single course. Our goal is to provide students with a solid, broad basis on which to build in later experiences, and to enable real understanding of a complex topic for which students realize there is a much greater depth of understanding that could be reached.

The disciplinary mix of students in the course is usually an even mix of information science and computer science students, with slightly fewer linguistics majors. Recently the Linguistics Department has established a concentration in Information Representation and Retrieval, for which the NLP course is a required course. Also, the course is cross-listed as an elective for computer science graduate students. All of the above facts contribute to the widely diverse mix of students in the NLP course, and has required us to develop a curriculum that enables all students to be successful in achieving solid competency in NLP.

4 Topics Covered

The topics in the course include typical ones covered in most NLP courses and are organized around the levels of language processing and the specific computational techniques within each of these. Discussions of more general theoretic notions such as statistical vs. symbolic NLP, representation theories, and language modeling are interspersed. A single example of topics that are taught within the levels of language processing include:

Morphology - Finite state automata
Lexicology - Part-of-speech tagging
Syntax - Parsing with context free grammars
Semantics - Word sense disambiguation
Discourse - Sublanguage analysis
Pragmatics - Gricean Maxims

Each of the topics has assigned readings, from the course's textbook, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* by Daniel Jurafsky & James H. Martin, as well as from recent and seminal papers.

5 Methods

What really enables the students to fully grasp the content of the course are the three important hands-on features of the course, namely:

1. Small, in-class group simulations of computational processes.
2. Team posters and public presentations reporting on the state-of-the-art in commercial NLP applications such as summarization, text mining, machine translation, question answering, speech recognition, and natural language generation.
3. Team projects implementing various levels of human language processing using open-source software on large collections.

Each of these features of the course is described in some detail in the following sections.

The course is designed around group projects, while the membership of the teams changes for each assignment. This is key to enabling a diverse group to learn to work with students from different disciplines and to value divergent experience. It has also proven extremely successful in forming a class that thinks of itself as a community and in encouraging sharing of best practices so that everyone advances their learning significantly further than if working alone or with the same team throughout the course. The way that teams are formed for the three types of projects varies, and will be described in each of the following three sections.

Furthermore, constant, frequent presentations to the class of the group work, no matter how brief, enable students to own their newly-gained understandings. In fact, this course no longer requires any written papers, but instead focuses on application of what is learned, first at the specific level of language processing, then to new data for new purposes, and then, to understanding real-world NLP systems performing various applications – with the group constantly reporting their findings back to the class.

5.1 In-class Group Simulations of Computational Processes

During the first third of the course, lectures on each level of language processing are followed by a 30 to 45 minute exercise that enables the students who work in small groups to simulate the process they have just learned about, i.e. morphological analysis, part-of-speech tagging, or parsing some sample sentences with a small grammar. These groups are formed by the professor in an ad hoc manner by counting off by 4 in a different pattern each week to ensure that students work with students on the other side of the room, given that friends or students from the same school tend to sit together. After the exercise, each group has 5 minutes to report back to the class on how they approached the task, with visuals.

We've found that the formation of these small groups is pedagogically sound and enables learning in three ways. First, the groups break down social barriers and as the course advances the students find it much easier to work together and are more comfortable in sharing their work. Secondly, the students begin to understand and value what the students from different disciplines bring to bear on NLP problems. That is, the computer scientists recognize the value of the deeper understanding of language of the linguistic students, and the linguistic students learn how the computer science students approach the task computationally. Thirdly, while there were concerns on our part that these simulations might be too easy, the students have affirmed in mid-term course evaluations (which are not required, but do provide invaluable insight into a class's engagement with and assimilation of the material) that these simulations really help them to understand conceptually what the task is and how it might be accomplished before they have to automate the processes.

5.2 Real World Applications of NLP

This year, two semester-long team projects were assigned – the usual team-based computer implementation of NLP for a particular computational task – and an investigation into how NLP is utilized in various state-of-the-art commercial NLP applications. The motivation for adding this second semester-long team project was that a number of the students in the course, particularly the masters students in Information Management, are most likely to encounter NLP in their work world when they need to advise on particular language-based applications. It has become clear, however, that as a result of this assignment, all of the students are quite pleased with their own improved ability to understand what a language-based technology is actually doing. Even if a student is more research-focused, they are intrigued by what might be done to improve or add to a particular technology.

Students are given two weeks to familiarize themselves outside of class with the suggested applications sufficiently to select a topic of real interest to them. This year's choices included Spell Correction, Machine Translation, Search Engines, Text Mining, Summarization, Question Answering, Speech Recognition, Cross-Language Information Retrieval, Natural Language Generation, and Dialogue Agents.

Students then sign up, on a first-come basis, for their preferred application. The teams are kept small (up to four) to ensure that each student contributes. At times a single student is sufficiently interested in a topic that a team of one is formed. Students arrange their own division of labor. There are three 10 to 20 minute report-backs by each team over the course of the semester, the first two to the class and the final one during an open invitation, school-wide Poster & Reception event. There are guidelines for each of the three presentations, as well as a stated expectation that the teams actively critique and comment on the presentations, both in terms of the information presented as well as presentational factors. Five minutes are allowed for class comments and students are graded on how actively they participate and provide feedback.

The 1st presentation is a non-technical overview of what the particular NLP application does and includes examples of publicly available systems / products the class might know. The 2nd presentation covers technical details of the application, concentrating on the computational linguistic aspects, particularly how such an application typically works, and the levels of NL processing that are involved (e.g., lexical, syntactic, etc). The 3rd presentation involves a poster which incorporates the best of their first two presentations and suggestions from the class, plus a laptop demo if possible.

As stated above, the 3rd presentation is done in an open school-wide Poster and Reception event which is attended by faculty and students, mainly PhD students. The Poster Receptions have proven very successful along multiple dimensions – first, the students take great pride in the work they are presenting; second, posters are better than one-time, in-class presentations as the multiple opportunities to explain their work and get feedback improve the students' ability to create the best presentation of their work; third, the wider exposure of the field and its applications builds an audience for future semesters and instills in the student body a sense of the reach and importance of NLP.

5.3 Hands-On NL Processing of Text

The second of the semester-long team projects is the computer implementation of NLP. The goal of the project is for students to gain hands-on experience in utilizing NLP software in the context of accomplishing analysis of a large, real-world data set. The project comprises two tasks, each of which is reported back to the class by each team. These presentations were not initially in the syllabus, but interestingly, the students requested that each team present after each task so that they could all learn from the experiences of the other teams.

The corpus chosen was the publicly available Enron email data set, which consists of about 250,000 unique emails from 150 people. With duplication, the data has approximately 500,000 files and takes up 2.75 gigabytes. The data set was prepared for public release by William Cohen at CMU and, available at http://www-2.cs.cmu.edu/~enron/. This data set is useful not only as real text of the email genre, but it can be easily divided into smaller subsets suitable for student projects. (And, of course, there is also the human interest factor in that the data set is available due to its use in the Enron court proceedings!)

The goal of the project is to use increasing levels of NLP to characterize a selected subset of Enron email texts. The project is designed to be carried out in two parts, involving two assigned levels of NLP. The first level, part-of-speech tagging, is accomplished as Task 1 and the second, phrase-bracketing or chunk-parsing, is assigned as Task 2. However, the overall characterization of the text is left open-ended, and the student teams chose various dimensions for their analyses. Projects included analyzing the topics of the emails of different people, social network analyses based on people and topics mentioned in the email text, and analyses based on author and recipient header information about each email.

Teams are established for these projects by the professor based on the capabilities and interests of the individual students as reported in short self-surveys. This resulted in teams on which there is a mix of computer science, linguistics and information science expertise. The teams accomplished the tasks of choosing a data analysis method, processing data subsets, designing NL processing to accomplish the analysis, programming the NL processing, conducting the data analysis, and preparing the in-class reports.

5.3.1 Tools Used in the Project

For preliminary processing of the Enron email files, programs and data made available by Professor Andrés Corrada-Emmanuel at the University of Massachusetts at Amherst, and available at http://ciir.cs.umass.edu/~corrada/ were used. The emails were assigned MD5-digest numbers in order to identify them uniquely, and the data consisted of mappings from the digest numbers to files, as well as to authors and recipients of the email. The programs contained filters that could be used to remove extraneous text such as headers and forwarded text. The teams adapted parts of these programs to convert the email files to files with text suitable for NL processing.

For the NL processing, the Natural Language Toolkit (NL Toolkit or NLTK), developed at the University of Pennsylvania by Loper and Bird (2002), and available for download from SourceForge at http://nltk.sourceforge.net/ was used. The NL Toolkit is a set of libraries written in the Python programming language that provides core data types for processing natural language text, support for statistical processing, and a number of standard processing algorithms used in NLP, including tokenization, part of speech (POS) tagging, chunk parsing, and syntactic parsing. The toolkit provides demonstration packages, tutorials, example corpora and documentation to support its use in educational classes. Experience using the Toolkit shows that in order to use the NL Toolkit, one member of each team should have at least some programming background in order to write Python programs that use the NL Toolkit libraries. The use of Python as the programming language was successful in that the level needed to use the NL Toolkit was manageable by the students with only a little programming background and in that the computer science students were able to adapt to the Python programming style and could easily utilize the classes and libraries.

At the beginning of the term project, the students were offered a lab session and lab materials to get them started. Since no one knew the Python programming language at the outset, there was an initial learning curve for the Python language as well as for the NL Toolkit. The lab materials provided to the students consisted of installation instructions for Python and NL Toolkit and a number of example programs that combined programming snippets from the NL Toolkit tutorials to process text through the NLP phases of tokenization, POS tagging and the construction of frequency distributions over the POS tagged text. During the lab session, some of the example programs were worked through as a group with the goal of enabling the students to become competent in Python and to introduce them to the NL Toolkit tutorials that had additional materials. The NL Toolkit tutorials are extensive on the lower levels of NL processing (e.g. lexical and syntactic) and students with some programming background were able to utilize them.

As part of their first task, the student teams were asked to select a subset of the Enron emails to work with. The entire Enron email directories were placed on a server for the teams to look at in making their selections. The teams also used information about the Enron employees as described in a paper by Corrada-Emmanuel (2005). Some student teams elected to work with different email topic folders for one person, while others chose a few email folders each from a small number of people (2-5). Their selected emails first needed to be processed to text using programs adapted from Corrada-Emmanuel. For the most part, the subcorpora choices of the student teams worked out well in terms of size and content. Several hundred emails turned out to be a good size, providing enough data to experience the challenges of long processing times and to appreciate why NLP is useful in processing large amounts of data, without being unduly overwhelmed. Initially, one team chose all the emails from several people. The number of email files involved was several thousand and it took several hours to unzip those directories, let alone process them, and they subsequently reduced the number of files for their analysis.

The first task was to analyze the chosen emails based solely on lexical level information, namely words with POS tags. NL Toolkit provides libraries for tokenization where the user can define the tokens through regular expressions, and the students used these to tailor the tokenization of their emails. The Toolkit also provides a regular expression POS tagger as well as n-gram taggers, and the students used these in combination for their POS tagging. Students experimented with the Brown corpus and a part of the Penn Treebank corpus, provided by NL Toolkit to train the POS taggers, and compared the results.

Building on the first task, the second task extended the analysis of the chosen emails to phrases from the text. Again, NL Toolkit provides a library for chunk parsing where regular expressions can be used to specify patterns of words with POS tags either to be included or excluded from phrases. Since chunk parsing depends on POS tagging, there was a need for a larger training corpus. A research center within the Information School has a license for Penn Treebank, and provided additional Penn Treebank files for the class to use for that purpose. Most teams used regular expressions to bracket proper names, minimal noun phrases, and verb phrases. One team used these to group maximal noun phrases, and another team used regular expressions to find patterns of communication verbs for use in social network analysis.

In retrospect, it was found that the chunk parsing did not take the teams far enough in NLP analysis of text. Experience in teaching using the NL Toolkit suggests that use of the syntactic parsing libraries to find more complex structures in the text would have provided more depth of analysis. Students also suggested that they would have liked to incorporate semantic level capabilities, such as the use of WordNet to find conceptual groupings via synonym recognition. The next offering of the course will include these improvements.

Using the NL Toolkit for NL processing worked out well overall and enabled the students to observe and appreciate details of the processing steps without having to write a program for every algorithm themselves. The tutorials are good, both at explaining concepts and providing programming examples. There were a few places where some data structure details did not seem to be sufficiently documented, either in the tutorials or in the API. This was true for the recently added Brill POS tagger, and is likely due to its recency of addition to the toolkit. However for the most part, the coverage of the documentation is impressive.

6 Evaluation

Multiple types of evaluation are associated with the course. First, the typical evaluation of the students by the professor (here, 2 professors) was done on multiple dimensions that contributed proportionately to the student's final grade as follows:

In-Class group exercises	20%
NLToolkit Team Assignments	35%
NLP Application Team Poster & Presentations	35%
Contributions to class discussion (both quality and quantity)	10%

Additionally, each team member evaluated each of their fellow team members as well as themselves. This was done for both of the teams in which a student participated. For each team member, the questions covered: the role or tasks of the student on the project; an overall performance rating from 1 for POOR to 4 for EXCELLENT; the rationale for this score, and finally; what the student could have done to improve their contribution. Knowledge of this end-of-semester team self-evaluation tended to ensure that students were active team contributors.

The professor was also evaluated by the students. And while there are quantitative scores that are used by the university for comparison across faculty and to track individual faculty improvements over time, the most useful feature of the student evaluations is the set of open-ended questions concerning what worked well in the course, what didn't work well, and what could be done to improve the course. Over the years of teaching this course, these comments (plus the mid-term evaluations) have been most instructive in efforts to find ways to improve the course. Frequently the suggestions are very practical and easy to implement, such as showing a chart with the distribution of grades on each assignment when they are returned so that the students know where they stand relative to the class as grading is on a scale of 1 to 10.

7. Indicators of Success

Finally, how is the success of this course measured in the longer term? For this, success is measured by: whether students elect to do continued work in NLP, either in the context of further courses in which NLP is utilized, such as Information Retrieval or Text Mining; whether the masters (and undergraduate) students decide to pursue an advanced degree based on the excitement engendered and knowledge gained from the NLP course; or whether PhD students elect to do continued research either in the school's Center for Natural Language Processing or as part of their dissertation. For students in a terminal degree program, success is reflected by their seeking and obtaining jobs that utilize the NLP they have learned in the course and that has provided them with a solid, broad basis on which to build. For several of the undergraduate computer science students in the course, their NLP experience has given them an added dimension of specialization and competitive advantage in a tight hiring market.

An additional measure of success was the request by the doctoral students in the home school for a PhD level seminar course to build on the NLP course. This course is entitled Content Analysis Research Using Natural Language Processing and will enable PhD students doing social science research on large textual data sets to explore and apply the NLP tools that are developed within the school, as well as to understand how these NLP tools can be successfully interleaved with commercial content analysis tools to support rich exploration of their data. As is the current course, this seminar will be open to PhD students from all schools across campus and already has enrollees from public policy, communications, and management, as well as information science.

8. Summary

While it might appear that a disproportionate amount of thought and attention is given to the more human and social aspects of designing and conducting this course, experience shows that such attention is the key to the success of this diverse body of students in learning and understanding the content of the course. Furthermore, given the great diversity in class-level and disciplinary background of students, this attention to structuring the course has paid off in the multiple ways exemplified above. While it is obvious that a course for computer-science majors alone would be designed quite differently, it would not provide the enriched understanding of the field of NLP and its application value that is possible with the contributions by the variety of disciplines brought together in this course.

Acknowledgements

We would like to acknowledge the contributions of the students in all the classes over the years whose efforts and suggestions have continually improved the course. We would to especially acknowledge this year’s class, who were especially contributory of ideas for improving and building on a currently successful course, namely Agnieszka Kwiatkowska, Anatoliy Gruzd, Carol Schwartz, Cun-Fang Cheng, Freddie Wade, Joshua Legler, Keisuke Inoue, Matthew Wolf, Michael Fudge, Michel Tinuiri, Olga Azarova, Rebecca Gilbert, Shuyuan Ho, Tuncer Can, Xiaozhony Liu, and Xue Xiao.

References

Loper, E. & Bird, S., 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: Association for Computational Linguistics.

Corrada-Emmanuel, A. McCallum, A., Smyth, P., Steyvers, M. & Chemudugunta, C., 2005. Social Network Analysis and Topic Discovery for the Enron Email Dataset. In Proceedings of the Workshop on Link Analysis, Counterterrorism and Security at 2005 SIAM International Conference in Data Mining.

Author Index