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.