Combining Open-Source with Research to Re-engineer a Hands-on Introductory NLP Course

Nitin Madnani **Bonnie J. Dorr**
Laboratory for Computational Linguistics and Information Processing
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland, College Park
{nmadnani,bonnie}@umiacs.umd.edu

Abstract

We describe our first attempts to re-engineer the curriculum of our introductory NLP course by using two important building blocks: (1) Access to an easy-to-learn programming language and framework to build hands-on programming assignments with real-world data and corpora and, (2) Incorporation of interesting ideas from recent NLP research publications into assignment and examination problems. We believe that these are extremely important components of a curriculum aimed at a diverse audience consisting primarily of first-year graduate students from both linguistics and computer science. Based on overwhelmingly positive student feedback, we find that our attempts were hugely successful.

1 Introduction

Designing an introductory level natural language processing course for a class of first year computer science and linguistics graduate students is a challenging task. It is important to strive for balance between breadth and depth—it is important not only to introduce the students to a variety of language processing techniques and applications but also to provide sufficient detail about each. However, we claim that there is another important requirement for a successful implementation of such a course. Like any other graduate-level course offered to first year students, it should encourage them to approach solutions to problems as researchers. In order to meet such a requirement, the course should have two important dimensions:

1. Access to a programming framework that provides the tools and data used in the real world so as to allow the students to explore each topic hands-on and easily attempt creative solutions to problems. The framework should be simple enough to use so that students are not bogged down in its intricacies and can focus on the course concepts.

2. Exposure to novel and innovative research in each topic. One of the most valuable contributions of a large community, such as the NLP and CL community, is the publicly accessible repository of research publications for a range of topics. While the commonly used textbooks describe established and mainstream research methods for each topic in detail, more recent research papers are usually omitted. By using such papers as the bases for programming assignments—instantiated in the framework described earlier—and exam questions, students can gain important insights into how new solutions to existing problems are formulated; insights that can only come from a hands-on approach to problem solving.

In this paper, we describe our attempts to engineer such a course. In section 2, we describe the specific goals we had in mind for such a course and how it differs from the previous version of the introductory course we taught at our institution. Section 3 discusses how we fully integrated an open-source programming framework into our curriculum and used it for programming assignments as well as in-class

Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics (TeachCL-08), pages 71–79,
Columbus, Ohio, USA, June 2008.

sessions. In a similar vein, section 4 describes our preliminary efforts to combine interesting research ideas for various topics with the framework above. We also have definite plans to expand the course curriculum to take more novel ideas from recent NLP literature for each topic and adapt them to instructive hands-on assignments. Furthermore, we are developing extensions and add-ons for the programming framework that we plan to contribute to the project. We outline these plans in section 6 and conclude in section 7.

2 Goals

We wanted our course curriculum to fulfill some specific goals that we discuss below, provide motivation wherever appropriate.

- **A Uniform Programming Framework**. The previous version of our introductory course took a more fragmented approach and used different programming languages and tools for different assignments. For example, we used an in-house HMM library written in C for any HMM-based assignments and Perl for some other assignments. As expected, such an approach requires students to familiarize themselves with a different programming interface for each assignment and discourages students to explore on their own. To address this concern, we chose the Python (Python, 2007) programming language and the Natural Language Toolkit (Loper and Bird, 2002), written entirely in Python, for all our assignments and programming tasks. We discuss our use of NLTK in more detail in the next section.

- **Real-world Data & Corpora**. In our previous course, students did not have access to any of the corpora that are used in actual NLP research. We found this to be a serious shortcoming and wanted to ensure that our new curriculum allowed students to use real corpora for evaluating their programming assignments.

- **Exposure to Research**. While we had certainly made it a point to introduce recent research work in our lectures for all topics in the previous course, we believed that a much richer integration was required in order to allow a more realistic peek into NLP research.

- **Satisfying a Diverse Audience**. We wanted the curriculum to appeal to both computer science and linguistics students since they the course was cross-listed in both departments.

- **Continuing Interest**. A large number of the students enrolled in the course were undecided about what research area to pursue. We wanted to present a fair picture of what NLP research actually entails and encourage any interested students to take the more advanced part of the course being offered later in the year.

3 Incorporating Open Source

We use the Python programming language and NLTK as our programming framework for the curriculum. Python is currently one of the most popular programming languages—it is fully object oriented and multi-platform, natively supports high-level dynamic data types such as lists and hashes (termed *dictionaries* in Python), has very readable syntax and, most importantly, ships with an extensive standard library for almost every conceivable task. Although Python already has most of the functionality needed to perform very simple NLP tasks, its still not powerful enough for most standard ones. This is where the Natural Language Toolkit (NLTK) comes in. NLTK[1], written entirely in Python, is a collection of modules and corpora, released under an open-source license, that allows students to learn and conduct research in NLP (Bird et al., 2008). The most important advantage of using NLTK is that it is entirely self-contained. Not only does it provide convenient functions and wrappers that can be used as building blocks for common NLP tasks, it also provides raw and pre-processed versions of standard corpora used frequently in NLP literature. Together, Python and NLTK constitute one of the most potent tools for instruction of NLP (Madnani, 2007) and allow us to develop hands-on assignments that can appeal to a broad audience including both linguistics and computer science students.

[1] `http://nltk.org`

```
Query: economy

Clinton 76 ############################################################################
 Carter 21 ####################
 GWBush 61 ############################################################
 Reagan 48 ################################################
   Bush 15 ###############
  Nixon 12 ############

Query: aid

Clinton  2 ##
 Carter  1 #
 GWBush  5 #####
 Reagan  9 #########
   Bush  2 ##
  Nixon  7 #######
```

Figure 1: An Excerpt from the output of a Python script used for an in-class exercise demonstrating the simplicity of the Python-NLTK combination.

In order to illustrate the simplicity and utility of this tool to the students, we went through an in-class exercise at the beginning of the class. The exercise asked the students to solve the following simple language processing problem:

Find the frequency of occurrences of the following words in the state-of-the-union addresses of the last 6 American Presidents: war, peace, economy & aid. Also draw histograms for each word.

We then went through a step-by-step process of how one would go about solving such a problem. The solution hinged on two important points:

(a) NLTK ships with a corpus of the last 50 years of state-of-the-union addresses and provides a native conditional frequency distribution object to easily keep track of conditional counts.

(b) Drawing a histogram in Python is as simple as the statement `print '#'*n` where n is the count for each query word.

Given these two properties, the Python solution for the problem was only 20 lines long. Figure 1 shows an excerpt from the output of this script. This exercise allowed us to impress upon the students that the programming framework for the course is simple and fun so that they may start exploring it on their own. We describe more concrete instances of NLTK usage in our curriculum below.

3.1 HMMs & Part-of-speech Tagging

Hidden Markov Models (Rabiner, 1989) have proven to be a very useful formalism in NLP and have been used in a wide range of problems, e.g., parsing, machine translation and part-of-speech (POS) tagging. In our previous curriculum, we had employed an in-house C++ implementation of HMMs for our assignments. As part of our new curriculum, we introduced Markov models (and HMMs) in the context of POS tagging and in a much more hands-on fashion. To do this, we created an assignment where students were required to implement Viterbi decoding for an HMM and output the best POS tag sequence for any given sentence. There were several ways in which NLTK made this extremely simple:

- Since we had the entire source code of the HMM module from NLTK available, we factored out the part of the code that handled the HMM training, parameterized it and provided that to students as a separate module they they could use to train the HMMs. Such refactoring not only allows for cleaner code boundaries but it also allows the students to use a variety of training parameters (such as different forms of smoothed distributions for the transition and emission probabilities) and measure their effects with little effort. Listing 1 shows how the refactoring was accomplished: the training code was put into a separate module called `hmmtrainer` and automatically called in the

Listing 1: A skeleton of the refactored NLTK HMM code used to build a hands-on HMM assignment

```
import hmmtrainer
import nltk.LidStoneProbDist as lidstone
class hmm:
  def __init__(self):
    params = hmmtrainer.train(smooth=lidstone)
    self.params = params

  def decode(self, word_sequence)

  def tag(self, word_sequence)
```

Listing 2: An illustration of the simple finite state transducer interface in NLTK

```
from nltk_contrib.fst import fst
f = fst.FST('test') #instantiate
f.add_state('1') # add states
f.add_state('2')
f.add_state('3')
f.initial_state = 1 # set initial
f.set_final('2') # set finals
f.set_final('3')
f.add_arc('1','2','a','A') # a -> A
f.add_arc('1','3','b','B') # b -> B
print f.transduce(['a','a','b','b'])
```

main `hmm` class when instantiating it. The students had to write the code for the `decode` and `tag` methods of this class. The HMM training was setup to be able to use a variety of smoothed distributions, e.g. Lidstone, Laplace etc., all available from NLTK.

- NLTK ships with the tokenized and POS tagged version of the Brown corpus—one of the most common corpora employed for corpus linguistics and, in particular, for evaluating POS taggers. We used Section A of the corpus for training the HMMs and asked the students to evaluate their taggers on Section B.

Another advantage of this assignment was that the if students were interested in how the supervised training process actually worked, they could simply examine the `hmmtrainer` module that was also written entirely in Python. An assignment with such characteristics in our previous course would have required knowledge of C++, willingness to wade through much more complicated code and would certainly not have been as instructive.

3.2 Finite State Automata

Another topic where we were able to leverage the strengths of both NLTK and Python was when introducing the students to finite state automata. Previously, we only discussed the fundamentals of finite state automata in class and then asked the students to apply this knowledge to morphological parsing by using PC-Kimmo (Koskenniemi, 1983). However, working with PC-Kimmo required the students to directly fill entries in transition tables using a very rigid syntax.

In the new curriculum, we could easily rely on the finite state module that ships with NLTK to use such automata in a very natural way as shown in Listing 2. With such an easy to use interface, we could concentrate instead on the more important concepts underlying the building and cascading of transducers to accomplish a language processing task.

As our example task, we asked the students to implement the Soundex Algorithm, a phonetic algorithm commonly used by libraries and the Census Bureau to represent people's names as they are pronounced in English. We found that not only did the students easily implement such a complex transducer, they also took the time to perform some analysis on their own and determine the shortcomings of the Soundex algorithm. This was only possible because of the simple interface and short development cycle provided by the Python-NLTK combination. In addition, NLTK also provides a single method[2] that can render the transducer as a postscript or image file that can prove extremely useful for debugging.

In our new version of the course, we consciously chose to use primarily open-source technologies in the curriculum. We feel that it is important to say a few words about this choice: an open-source project

[2]This method interfaces with an existing installation of *Graphviz*, a popular open-source graph drawing software (Ellson et al., 2004).

not only allows instructors to examine the source code and re-purpose it for their own use (as we did in section 3.1) but it also encourages students to delve deep into the programming framework if they are curious about how something works. In fact, a few of our students actually discovered subtle idiosyncrasies and bugs in the NLTK source while exploring on their own, filed bug reports where necessary and shared the findings with the entire class. This experience allowed all students to understand the challenges of language processing.

More importantly, we believe an open-source project fosters collaboration in the community that it serves. For example, a lot of the functionality of NLTK hinges on important technical contributions, such as our SRILM interface described in section 6, from the large academic NLP community that can be used by any member of the community for research and for teaching.

4 Incorporating Research

Besides employing a uniform programming framework that the students could pick up easily and learn to explore on their own, the other important goal of the new curriculum was to incorporate ideas and techniques from interesting NLP research publications into assignments and exams. The motivation, of course, was to get our students to think about and possibly even implement these ideas. Since we cannot discuss all instances in the curriculum where we leveraged research publications (due to space considerations), we only discuss two such instances in detail below.

The first topic for which we constructed a more open-ended research-oriented assignment was lexical semantics. We focused, in particular, on the WordNet (Fellbaum, 1998) database. WordNet is a very popular lexical database and has been used extensively in NLP literature over the years. In the previous course, our assignment on lexical semantics asked the students to use the online interface to WordNet to learn the basic concept of a *synset* and the various relations that are defined over synsets such as hyponymy, hypernymy etc. A very simple change would have been to ask the students to use the WordNet interface included with NLTK to perform the same analysis. However, we thought that a more interesting assignment would be to explore the structure of the four WordNet taxonomies (Noun, Verb, Adjective and Adverb). This taxonomy can be simplified and thought of as a directed acyclic graph $G = (V, E)$ where each synset $u \in V$ is a node and each edge $(u, v) \in E$ represents that v is a hypernym of u. Given such a graph, some very interesting statistics can be computed about the topology of WordNet itself (Devitt and Vogel, 2004). In our assignment, we asked the students to use the NLTK WordNet interface to compute some of these statistics automatically and answer some interesting questions:

(a) What percentage of the nodes in the Noun taxonomy are leaf nodes?

(b) Which are the nine most general root nodes in the Noun taxonomy and what is the node distribution across these roots?

(c) Compute the *branching factor* (number of descendants) for each node in the Noun taxonomy both including and excluding leaf nodes. What percentage of nodes have a branching factor less than 5? Less than 20? Does this tell something you about the shallowness/depth of the taxonomy?

(d) If we plot a graph with the number of senses of each verb in the Verb taxonomy against its polysemy rank, what kind of graph do we get? What conclusion can be drawn from this graph?

(e) Compare the four taxonomies on average polysemy, both including and excluding monosemous words. What conclusions can you draw from this?

Of course, the assignment also contained the usual questions pertaining to the content of the WordNet database rather than just its structure. We believe that this assignment was much more instructive because not only did it afford the students a close examination into the usage as well as structure of a valuable NLP resource, but also required them to apply their knowledge of graph theory.

The second instance where we used a research paper was when writing the HMM question for the final exam. We thought it would be illuminating to ask the students to apply what they had learned in class about HMMs to an instance of HMM used in an actual NLP scenario. For this purpose, we chose the HMM described in (Miller et al., 1999) and as shown in Figure 2. As part of the question, we ex-

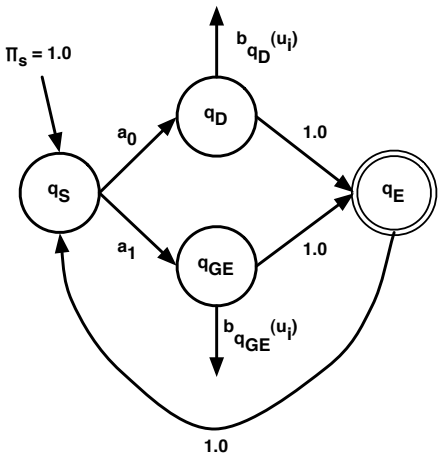

Figure 2: An HMM used and described in a popular research publication formed the basis of a question in the final exam.

plained the information retrieval task: generate a ranked list of documents relevant to a user query $U = \langle u_i \rangle$, where the rank of the document D is based on the probability $P(D \text{ is relevant}|U)$. We further explained that by applying Bayes' theorem to this quantity and assuming a uniform prior over document selection, the only important quantity was the probability of the query U being generated by a relevant document D, or $P(U|D \text{ is relevant})$. The rest of the question demonstrated how this generative process could be modeled by the HMM in Figure 2:

- Start at the initial state q_S.
- Transition with the probability a_0 to state q_D which represents choosing a word directly from document D OR transition with probability a_1 to state q_{GE} which represents choosing a word from "General English", i.e., a word unrelated to the document but that occurs commonly in other queries.
- If in state q_D, emit the current, say i^{th}, query word either directly from document D with emission probability $b_{q_D}(u_i)$. Otherwise, if in state q_{GE}, emit the current query word from "General English" with emission probability $b_{q_{GE}}(u_i)$.
- Transition to the end state q_E.
- If we have generated all the words in the query, then stop here. If not, transition to q_S and repeat.

Given this generative process, we then asked the students to answer the following questions:

(a) Derive a simplified closed-form expression for the posterior probability $P(U|D \text{ is relevant})$ in terms of the transition probabilities $\{a_0, a_1\}$ and the emissions probabilities $\{b_{q_D}(u_i), b_{q_{GE}}(u_i)\}$. You may assume that $U = \langle u_i \rangle_{i=1}^{n}$.

(b) What HMM algorithm will you use to compute $P(U|D \text{ is relevant})$ when implementing this model?

(c) How will you compute the maximum likelihood estimate for the emission probability $b_{q_D}(u_i)$?

(d) What about $b_{q_{GE}}(u_i)$? Is it practical to compute the actual value of this estimate? What reasonable approximation might be used in place of the actual value?

This question not only required the students to apply the concepts of probability theory and HMMs that they learned in class but also to contemplate more open-ended research questions where there may be no one right answer.

For both these and other instances where we used ideas from research publications to build assignments and exam questions, we encouraged the students to read the corresponding publications after they had submitted their solutions. In addition, we discussed possible answers with them in an online forum set up especially for the course.

5 Indicators of Success

Since this was our first major revision of the curriculum for an introductory NLP course, we were interested in getting student feedback on the changes that we made. To elicit such feedback, we designed a survey that asked all the students in the class (a total of 30) to rate the new curriculum on a scale of one to five on various criteria, particularly for the experience of using NLTK for all programming assignments and on the quality of the assignments themselves.

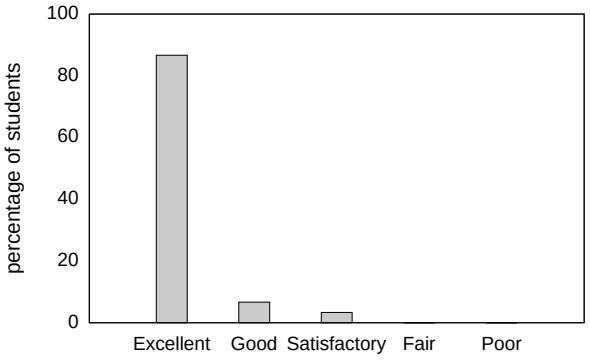

Figure 3: Histogram of student feedback on the experience of using the Python-NLTK combination.

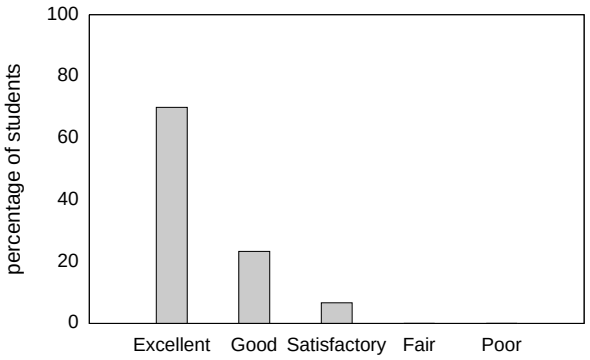

Figure 4: Histogram of student feedback on the quality of course assignments.

Figures 3 and 4 show the histograms of the students' survey responses for these two criteria. The overwhelmingly positive ratings clearly indicate that we were extremely successful in achieving the desired goals for our revised curriculum. As part of the survey, we had also asked the students to provide any comments they had about the curriculum. We received a large number of positive comments some of which we quote below:

"Using Python and NLTK for assignments removed any programming barriers and enabled me to focus on the course concepts."

"The assignments were absolutely fantastic and supplemented the material presented in class."

"A great experience for the students."

The first comment—echoed by several linguistics as well as computer science students—validates our particular choice of programming language and framework. In the past, we had observed that linguistics students with little programming background spent most of their time figuring out how to wield the programming language or tool to accomplish simple tasks. However, the combination of Python and NLTK provided a way for them to work on computational solutions without taking too much time away from learning the core NLP concepts.

While it is clearly apparent to us that the students really liked the new version of the curriculum, it would also have been worthwhile to carry out a comparison of students' reviews of the old and new curricula. The most frequent comments that we saw in older versions of the course were similar to the following:

"Although I feel you did a decent job repeating and pointing out the interesting facts of the book, I don't think you really found many compelling examples of using these techniques in practice."

The feedback we received for the revamped curriculum, such as the second comment above, clearly indicated that we had addressed this shortcoming of the older curriculum. However, due to significant format changes in the review forms between various offerings of this course, it is not possible to conduct a direct, retroactive comparison. It is our intent to offer such comparisons in the future.

6 Future Plans

Given the success that we had in our first attempt to re-engineer the introductory NLP course, we plan to continue: (1) our hands-on approach to programming assignments in the NLTK framework and, (2) our practice of adapting ideas from research publications as the bases for assignment and examination problems. Below we describe two concrete ideas for the next iteration of the course.

1. **Hands-on Statistical Language Modeling**. For this topic, we have so far restricted ourselves to the textbook (Jurafsky and Martin, 2000); the in-class discussion and programming assignments have been missing a hands-on component. We have written a Python interface to the SRI Language Modeling toolkit (Stolcke, 2002) for use in our research work. This interface uses the Simplified Wrapper & Interface Generator (SWIG) to generate a Python wrapper around our C code that does all the heavy lifting via the SRILM libraries. We are currently working on integrating this module into NLTK which would allow all NLTK users, including our students in the next version of the course, to build and query statistical language models directly inside their Python code. This module, combined with the large real-world corpora, would provide a great opportunity to perform hands-on experiments with language models and to understand the various smoothing methods. In addition, this would also allow a language model to be used in an assignment for any other topic should we need it.

2. **Teaching Distributional Similarity**. The idea that a language possesses *distributional structure*—first discussed at length by Harris (1954)—says that one can describe a language in terms of relationships between the occurrences of its *elements* (words, morphemes, phonemes). The name for the phenomenon is derived from an element's *distribution*—sets of other elements in particular positions that occur with the element in utterances or sentences. This work led to the concept of *distributional similarity*—words or phrases that share the same distribution, i.e., the same set of words or in the same context in a corpus, tend to have similar meanings. This is an extremely popular concept in corpus linguistics and forms the basis of a large body of work. We believe that this is an important topic that should be included in the curriculum. We plan to do so in the context of lexical paraphrase acquisition or synonyms automatically from corpora, a task that relies heavily on this notion of distributional similarity. There has been a lot of work in this area in the past years (Pereira et al., 1993; Gasperin et al., 2001; Glickman and Dagan, 2003; Shimohata and Sumita, 2005), much of which can be easily replicated using the Python-NLTK combination. This would allow for a very hands-on treatment and would allow the students to gain insight into this important, but often omitted, idea from computational linguistics.

7 Conclusion

Our primacy goal was to design an introductory level natural language processing course for a class of first year computer science and linguistics graduate students. We wanted the curriculum to encourage the students to approach solutions to problems with the mind-set of a researcher. To accomplish this, we relied on two basic ideas. First, we used a programming framework which provides the tools and data used in the real world so as to allow hands-on exploration of each topic. Second, we adapted ideas from recent research papers into programming assignments and exam questions to provide students with insight into the process of formulating a solution to common NLP problems. At the end of the course, we asked all students to provide feedback and the verdict from both linguistics and computer science students was overwhelmingly in favor of the new more hands-on curriculum.

References

Steven Bird, Ewan Klein, Edward Loper, and Jason Baldridge. 2008. Multidisciplinary Instruction with the Natural Language Toolkit. In *Proceedings of the Third ACL Workshop on Issues in Teaching Computational Linguistics*.

Ann Devitt and Carl Vogel. 2004. The Topology of

WordNet: Some metrics. In *Proceedings of the Second International WordNet Conference (GWC2004)*.

J. Ellson, E.R. Gansner, E. Koutsofios, S.C. North, and G. Woodhull. 2004. Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools. In *Graph Drawing Software*, pages 127–148. Springer-Verlag.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

Caroline Gasperin, P. Gamallo, A. Agustini, G. Lopes, and Vera de Lima. 2001. Using syntactic contexts for measuring word similarity. In *Workshop on Knowledge Acquisition Categorization, ESSLLI*.

Oren Glickman and Ido Dagan. 2003. Identifying lexical paraphrases from a single corpus: A case study for verbs. In *Recent Advantages in Natural Language Processing (RANLP'03)*.

Zellig Harris. 1954. Distributional Structure. *Word*, 10(2):3.146–162.

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

Kimmo Koskenniemi. 1983. Two-level morphology: a general computational model for word-form recognition and production. Publication No. 11, University of Helsinki: Department of General Linguistics.

Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, pages 62–69.

Nitin Madnani. 2007. Getting Started on Natural Language Processing with Python. *ACM Crossroads*, 13(4).

D. R. Miller, T. Leek, and R. M. Schwartz. 1999. A hidden Markov model information retrieval system. In *Proceedings of SIGIR*, pages 214–221.

Fernando Pereira, Naftali Tishby, and Lillian Lee. 1993. Distributional clustering of english words. In *Proceedings of ACL*, pages 183–190.

Python. 2007. The Python Programming Language. `http://www.python.org`.

Lawrence R. Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Mitsuo Shimohata and Eiichiro Sumita. 2005. Acquiring synonyms from monolingual comparable texts. In *Proceedings of IJCNLP*, pages 233–244.

Andreas Stolcke. 2002. SRILM – an extensible language modeling toolkit. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*.