Massachvsetts Institvte of Technology
Department of Electrical Engineering and Computer Science
6.863J/9.611J, Natural Language Processing
Fall 2012, final project ideas

1 Final Project Ideas

This is a list of final project ideas, including pointers to projects from previous years, followed by topics one can do using NLTK tools, plus more, divided into subjects areas. We also list several of the most commonly available open-source NLP tool sites, along with a data source for many kinds of corpora and projects related to those corpora. Note: there are about 120 project suggestions in this list!

1.1 Tools and Corpora

Besides the NLTK tools and projects listed separately, you may want to take a look at the following sites for tools, corpora, and project ideas, as well programming methods you can borrow. ("Good artists copy, great artists steal" – Pablo Picasso; or, "the virtue of theft over honest toil" – Bertrand Russell.)

1. **Corpora**: `http://www.hit.uib.no/corpora/` has the most extensive list of corpora, questions about corpora, corpora tools, projects, and data sources. Search here first before you have any questions about corpora – most likely, it will have been answered before.

2. **Open source NLP tools**. Several good places, including those below (beyond NLTK itself). See the wiki at `http://en.wikipedia.org/wiki/List_of_natural_language_processing_toolkits` for an overview. There's OpenNLP, at `http://opennlp.apache.org/`; FreeLing at `http://nlp.lsi.upc.edu/freeling/`; LingPipe at `http://alias-i.com/lingpipe/`. LingPipe has very good tutorials, demos, and nice code on these topics, which you might re-purpose for a project:

   (a) Part of speech detection
   (b) Spelling correction
   (c) Database text mining
   (d) Interesting phrase detection (as in Amazon books); see `http://www.amazon.com/gp/search-inside/sipshelp.html`
   (e) Chinese word segmentation
   (f) Sentiment analysis
   (g) Expectation maximiation
   (h) Word sense disambiguation

   All of this is all in Java.

1.2 Projects from 2009-2011

We have posted all the writeups from these projects on the class website, as `web.mit.edu/6.863/fall2012/projects-2011/`. If you can make a good case for the re-use of the source code to *extend* a project in important ways, I am always open to persuasion. You will get a good sense of the diversity and range of possible projects from the topics just below.

1. Translating sentences to parses and then to SQL queries, with the "semantic" side being a database – then answering the queries, etc. (An *extremely* nice project to extend – built on top of the NLTK tools.) A terrific project. The code is available if you want to extend this in several directions. (I would be interested if some team wants to do this.)

2. Filtering parses with lexical semantics: Uses WordNet and FrameNet to filter out unlikely parses that are OK syntactically. Built using NLTK. For WordNet see here: `http://wordnet.princeton.edu/`

   For FrameNet (a set of "possible arguments" for verbs), along with 3 other very useful databases of verb-argument structure, see: `http://verbs.colorado.edu/verb-index/`

3. A complete analysis of Spanish morphology (word formation). This uses an existing NLTK subsystem we have build, called KIMMO; for further projects KIMMO see below, under *Morphology*. (One done in 2011 was by, ahem, our very own TA, Geza, on Japanese morphology!)

4. A morphology project for Turkish (a very interesting language).

5. A technical question answering Bot, for SIPB questions.

6. Examining and classifying the "color: metadata field of products listed for sale on ecommerce platforms (e.g., Amazon). (Using Amazon, Newegg, BestBuy listings.)

7. Improved comma analysis for statistical and other parsers.

8. Classification and summarization of legal cases. (This used an n-gram approach.)

9. Maximum-entropy part of speech tagging (that beats the other max-entropy methods), in NLTK.

10. Music improvisation using n-grams. (Yes, indeed.) (Uses data in `http://people.csail.mit.edu/yalesong/6.863-Music.Improviser/`.)

11. A stock market message classifier.

12. Japanese-English transliteration.

13. Corpora bias detection: how much will changing domains affect the performance of using a corpus not meant for that domain?

14. Classification and keyword extraction on Anne Hunter's "joblist" emails. (A nice idea.) Uses a good package for morphologically annotating words: `http://morphadorner.northwestern.edu/`.

15. A comparison of different methods to classify The Tech newspaper articles (using NLTK tools: naive Bayes; maximum entropy; multinomial) – straightforward, not so cutting-edge.

16. Link grammar: an alternative way to represent relationships among words, instead of context-free grammars. This project built a link-grammar parser and tested it against CFGs. (See Daniel Sleator and Davy Temperley. 1991. Parsing English with a Link Grammar. Carnegie Mellon University Computer Science technical report CMU-CS-91-196, October 1991.) It would be interesting to make this probabilistic.

17. An improved Bayesian part of speech tagger with constant-time performance (so good for devices with limited CPU/memory).

18. Comparison of two methods for extracting text information, feeding a support-vector machine classifier: one a conditional-random field named-entity recognition parser (based on the Stanford NER); the other a "region algebra" NER system.

19. A discourse-based parsing system (heavily modifying the NLTK existing parser plus adding a semantic interpreter), to answer questions in a dialog form.

20. A method to locate referred-to keywords, e.g., "it", in mobile phone text messages, for use in a Google-adword system. (E.g., "my computer broke." "Did you get it fixed?".)

21. Improving "Powerset"'s NLP retrieval system (note: Powerset was bought by Microsoft; this final project does better!) – retrieves and parses Wikipedia articles for summarization.

22. Event tracker: automatic classifier running off of the MIT Even Calendar that recognizes seminar and other events of interest, does NLP processing to alert user/fill in calendar, etc.

23. Parallelizing the CKY parser (also with other methods); done in MIT CILK system. (The CKY algorithm is a natural to speed up, since it is a matrix-based method. You can actually reduce it to matrix multiplication. Interestingly, there does not seem to be much effort in speeding up the treebank parsers this way.)

24. Semantic verb similarity (small project) using the VerbOcean network of verb similarities. See: `http://demo.patrickpantel.com/demos/verbocean/`.

25. A grammatical system for generating classical architectural building plans (definitely not the Stata center). Quite a sophisticated visual gui as well.

26. An extension of a simple semantic interpreter to handle negation in sentences.

27. Automatic newsgroup analysis.

1.3 Treebank related projects

We have both the full Penn Tree Bank (PTB) along with source code for most of the current statistically-trained parsers available for use: the Charniak-Johnson 2004 parser (in C); the Berkeley parser (java); the Stanford parser (java); and the Bikel/Collins parser (java). Finally, we have a so-called *categorial grammar* parser, called *C&C*. There are many interesting projects that can be done here, some along the lines of the lectures for the coming week:

1. Improve Prepositional Phrase attachment in treebank parsing by using WordNet, Propbank, or Framenet (see `http://verbs.colorado.edu/verb-index/` as per above).

2. Improve some aspect of the PTB where the annotation does not follow an adequate linguistic theory. For example, Noun Phrases in the PTB are "flat", but should be binary-branching. "Gaps" are annotated, but not used by the learning methods. Try re-training the Bikel/Collins parser on good NP structure (very fast to do), and see whether performance can be improved. Similarly, sentences with "gaps" cause most of the performance gaps (a pun) between PTB statistical parsers and accurate predicate-argument recovery. Focus on one aspect of this by "putting back" the phrases that are the "fillers" for some cases, and see whether you can boost performance. For this, we have a large sample of wh-question types sentences for use in training.

3. Fix the *read/red* bug discussed in lecture.

4. Run any of the statistical parsers on a sample of (English, or other) Wikipedia articles, as in the design discussed in lecture, but with some new linguistic phenomenon of your choice (you can discuss this with me for ideas). Analyze the performance.

5. Analyze to what extent errors are driven by incorrect part of speech tagging, and how this might be fixed.

We also have a treebank based on parents' speech to children (called CHILDES). You could use this in a number of ways.

1. Train the Bikel/Collins parser on the CHILDES corpus, and analyze the precision and recall of the results (making sure to divide the data into training, development, and test sets).

2. Improved part of speech tagging for CHILDES. It currently does not use anything but the PTB tags. Here also, one could either learn or improve the tags by using information from FrameNet or PropNet (see the url above).
3. Start with a very simple skeleton grammar (like initial step of the Berkeley parser, but simpler), and use this to bootstrap a learner on CHILDES. This would be a real model for child language acquisition.

Other past projects: please ask me if you'd like to do these, as I have writeups and code for them:

1. Learning Spanish morphology by Bayesian methods.
2. Intelligent pronoun resolution (finding antecedents of pronouns)
3. Italian-English machine translation, by full syntax and semantic interpretation
4. Parsing prescriptions for a medical database.

Other projects

NLTK lists 50 or so good projects, divided into 18 linguistically-oriented projects, and 36 NLTK-programming type projects. Their list is here: `http://ourproject.org/moin/projects/nltk/ProjectIdeas`. You should email/talk to me if you want to take up one of these.

Text analysis

1. Use a database of "sentiment" organized words, SentiWord Net along with NLTK tools to develop a sentiment analyzer. `http://sentiwordnet.isti.cnr.it/`. This page has links to many papers that have used this database. For data, there are many resources, a basic one is: `http://www.cs.cornell.edu/people/pabo/movie-review-data/`. A typical approach uses python and Naive Bayes classification; see if you can do better than this! (e.g., `http://smm.streamcrab.com,`).
2. In fact, sentiment analysis is a HUGE field; see `http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html` for an excellent source of sentiment parsers, books on sentiment analysis and so forth. See if you can reproduce what Microsoft does in BING search, and what Google does in Google search.
3. For twitter data on the same, which could be the foundation of a twitter-based project: `http://www.laurentluce.com/posts/twitter-sentiment-analysis-using-python-and-nltk/`.
4. Text summarization. Another large field; for an open-source version, see `http://libots.sourceforge.net/`. For a good list of nine such public projects, see: `http://web.science.mq.edu.au/~swan/summarization/projects_full.htm`
5. Topic detection. This has been the subject of several DARPA and other large scale funding. For a good set of corpora, see `http://projects.ldc.upenn.edu/TDT/`. If you can develop something beyond "standard" approaches that use clustering and basic machine learning methods, this can be fun.

Morphology projects

We have not covered "word parsing" this term, but as mentioned above we have an implemented system that is part of NLTK that does "parse words" (as demonstrated in lecture). That system, called KIMMO, could be extended in many interesting ways, both in terms of functionality and looking at different languages. You can read more about this topic in the first item suggested below.

1. For software you can run, as well as examples of lots of projects here, see this: `http://www.stanford.edu/~laurik/fsmbook/home.html`
2. Build a complete set of KIMMO rules to handle a language other than Spanish or Turkish, e.g., Portuguese, Hungarian, Japanese. (For Japanese, see Geza, who's done this).
3. Build a system to handle a non-concatenative language, like Arabic, Hebrew, etc. (Reference for Arabic: J. McCarthy's PhD. thesis at MIT, in the MIT Humanities Library). Note: in the past, people have implemented their own system (not Kimmo) to do this, in Scheme. I can provide you with basic skeleton code for this. It is quite challenging.
4. Implement finite-state rule compilation (i.e., combining the finite-state transducers from rules into one large one, via the methods of composition and/or intersection). This is challenging.

Part of speech tagging

1. From the NLTK docs on tagging: Use some of the estimation techniques in `nltk.probability`, such as Lidstone and Laplace estimation, to develop a statistical tagger that does a better job than n-gram backoff taggers in cases where contexts encountered during testing were not seen during tagging. Read up on the TnT tagger, since this provides the relevant technical background: `http://www.aclweb.org/anthology/A00-1031`.
2. Carry out experiments comparing taggers in morphologically rich languages like Estonian, Slovenian, Hungarian, etc.
3. Implement a maximum entropy tagger within NLTK and compare it to the backoff taggers. (Previous projects have done this, but perhaps you can do even better.) See: `http://acl.ldc.upenn.edu/W/W96/W96-0213.pdf.` There is a maxent classifier in nltk, but you would have to bend it to the tagging type design. This is also useful for text classification.
4. Improve the Brill transformational part of speech tagger by adding simple grammatical information from a parser, e.g., that a marker like "of" tells us that a Noun should follow. (This improves the Brill tagger a great deal, as it turns out.)
5. Modify the Brill tagger so that it can tag email. (You will need to possibly add new tags to handle the form of email.) In addition, map the resulting output into classification "bins" so that the email is classified as to its semantic (or syntactic) "type". (Developing a set of 'types' is important here.)
6. Investigate different "smoothing" methods for languages that vary greatly from English, that don't have as much hand-tagged data, and see how one could build adaptive tagging systems, bootstrapping from a core, small set of tagged sentences. You can use the smoothing methods in NLTK.

Parsing projects

(Some of these broadly echo previous projects, but they are still OK.)

1. (For syntax fans) Write a context-free grammar to handle all the roughly 100 distinct sentences in the original paper on generalized phrase structure grammar –either with or without features. The reference: "Unbounded dependencies and coordinate structure," *Linguistic Inquiry*,12, 155–184. This is quite a challenge (as I've mentioned in class), but very rewarding.
2. Examine the Collins parser in terms of the errors it makes with, e.g., conjunctions and mismatches with lexical-conceptual structure, and how these could be fixed by using FrameNet, PropNet, or WordNet.

3. Build a set of feature-based context-free rules that will handle Spanish, Italian, German, Hebrew, Gaelic, Russian, Serbian, or Esperanto sentences. Please include sentences including conjunctions (and discuss this with me in advance). (Yes, people have done Gaelic in the past.) (For Russian, see the last item – it is a kind of free word order language.)

4. Add a front end to an already-implemented Scheme chart parser that includes many lexical conceptual (semantic) items, here. You can (i) add a probabilistic component to the rules; or (ii) add a well-formed substring table to the method it uses for parsing. This method is based on very recent linguistic theories ("minimalism") – I can give you references on that, too. There are many ways that you could add a question-answering component to this parser. I can provide complete Scheme code for this parser.**Note: I am keenly interested in this, so would provide some extra support for any team here.**

5. Link grammars: Look at another method for parsing distinct from forming "tree structures" – e.g., "link grammar", which builds syntactic relations between words, rather than trees. Working parser for windoze/linux/etc is available, along with many details; see `http://www.link.cs.cmu.edu/link/`. Compare this to traditional context-free parsing for natural languages – which one works better? How could you tell?

6. Earley-parsing: this is a more top-down approach to CFG parsing (rather than bottom-up), that is often faster. Investigate optimizations/efficiency hacks for Earley parsing: precomputing chains of rules; lookahead; using an "ill-formed" substring table; matrix multiplication. I can give you a reference for all this from Graham, Harrison, and Ruzzo, 1978.

7. For psychologically oriented folks: Push deeper into the distinctions between shift-reduce/shift-shift conflicts as in Assignment 5, as psychologically valid models for human sentence processing. Implement the 2-stage Chomsky-Miller parser, based on intonational structure/shallow phrase breaks, followed by parse tree "readjustment." This is an important topic that has never been fully followed up in the published literature, as far as I know. I can supply you with references for the Chomsky-Miller model.

8. Parsing seminar announcements, mapping syntax to semantics. (This project used 3 types of parsing: extended regular expressions, shallow parsing, and full CFG parsing.)

9. Extend a feature-based parser so that it can be used in a "relaxation mode" for grammar checking (a la Microsoft Word).

10. Find some way to detect doubled words as being *correct* in some cases, e.g., *with with* is sometimes correct, but no method I know of will properly not flag such correct occurrences.

11. Build a parser that directly computes only precedence and dominance relations, rather than using context-free rules, and see if you can modify chart parsing to handle this. This would again be a novel result that could be published. It is much better for parsing conjunctions (sentences with *and* in them).

12. Parallel context-free parsing: using a chart parser, or parallel stacks. Fold probabilistic rules into the system. (Note that since parsing like this is, ultimately, akin to matrix multiplication, one can use any parallel speed-up method applicable to matrix multiplication as well.) The first task has been done by a previous group, so a new project would add probabilistic rules to such a system.

1.4 Semantics-oriented Projects

There are many ways to create a real semantic interpretation-db system – we list some of these projects here. (You have seen the semantics-sql translator in action – it would be good to add to that.)

1. Write a system using WordNet that can solve "analogy" problems in the GRE, for instance, "sentence completion" examples like the following. You can use WordNet and FrameNet to do this.

Vain and prone to violence, Caravaggio could not handle success: the more his (1)___as an artist increased, the more (2___ his life became.

Answer choices for question 2: Blank (1) temperance, notoriety, eminence; Blank (2), tumultuous, providential, dispassionate.

2. Use WordNet's "syngrams" to find suitable synonym choices for a word in text.

3. Add quantification (every, some, all, the, a, etc.) to the questions the semantic interpreter can currentl handle, making sure that you can return different answers in the case of scoping ambiguities, e.g *Did every guy sit on a park bench?* This requires the lambda calculus interpretation of quantifiers. Doing the scoping properly is very challening in general – I can give you examples.

4. Modify a db-semantic system so that it actually uses SQL. In addition – add new semantic extensions, e.g., the difference between intersective adjectives (like "big book" means big AND book), vs. non-intersective adjectives (like "fake book", which does not mean "fake" AND "book").

5. Add relative clauses to our NLTK semantic interpreter, along with rightward movement in the syntax, so the system can answer questions like *Which guy with red hair sat on a park bench* or *Which book was published yesterday about nuclear war.*

6. Add conjunctions. Note that the scope of conjunctions interacts with quantification, and you should take this into account: *John walks and talks* is logically equivalent to *John walks and John talks*, but *some guy walks and talks* is not equivalent to *some guy walks and some guy talks* (because we could be talking about two different guys in the second statement, but not the first.)

7. Find a suitable lambda-expression for the determiner *no*, and add it to our NLTK implementation. Test your solution with sentences like *No man walks.*

8. Add an ability to process multiple sentences, correctly handling backwards anaphora (*it*, pronouns, and definite Noun Phrases), as well as changes in focus. For example, in the sequence:

    Pick up the red block.

    Put it on the table (Now *It* refers to the red block. Thus the system should save the NP referent when it tries to carry out the next command. Add a data structure that can accomplish this. Note the following dialog:

    Put the blue block on the table.

    Put it on the red block. (*it* means block, not table)

    Move the ball onto it.

    Clear it. (still the blue block, not the ball)

    Put the ball on the blue block.

    Put it back on the table. (now *it* is the ball) See Brady and Berwick, Computational Models of Discourse, MIT Press, for more detail.

9. Lexical semantics. Look at the verbs in `evca.scm` and compare them to Wordnet. Then add a significant chunk of new verbs, taken from WordNet, in the style of the lambda forms in the web file, and integrate it into a parsing and semantic interpretation system.

10. Add a 'meaning postulate' system based on Wordnet's *synonym sets* to a list of basic verbs, to arrive at a reasoning component for semantic interpretation.

11. Add a representation of *time* so that a semantic/syntax system can answer when something happened as well as whether it happened before or after some other event. (One could simply add a new TIME field to an event representation, to start, but I want you do more.) You should add the Reichenbach representation of time – I can get you this reference – as a minimum. For a bigger challenge, add a reasoning component that would represent the temporal contiguity and continguity of events, so deducing the order in which events occur.