Massachvsetts Institvte of Technology
Department of Electrical Engineering and Computer Science
6.863J/9.611J, Natural Language Processing
Assignment 6: Statistical Parsing and Lexical Semantics
Handed out: November 14, 2012 **Due: November 28, 2012**
Last Updated: Tuesday 27th November, 2012 at 13:13
`http://web.mit.edu/6.863/www/fall2012/labs/assignment6.pdf`

Goals of the Assignment. This assignment explores the details about trained PCFGs for parsing, along with lexical (word) semantics.

In Part I, you will learn about the following:

1. What are the strengths and weaknesses of current state-of-the-art statistical natural language parsers? If they were perfect, we'd be done, at least for the syntax part of natural language processing. But the parsers are not perfect. We will look at ambiguity (again). One of the things modern statistical parsers do better is adding information about particular words in order to figure out how to prune parsing possibilities. Classic examples would be those like *I saw the guy on the hill with the telescope.* Does *with the telescope* associate more strongly with *the hill* or *saw*?

Part II will give deeper understanding of how statistical parsers work and how they interact with lexical frequencies, and syntactic and semantic regularities. In particular, you will be assigned a 'verb of your own' and then asked to:

2. Investigate the connection between lexical (word-level) semantics and parsing, using the Penn Tree Bank (PTB) as a concrete test bed.
3. Explore how a state-of-the-art probabilistic parser will handle these issues.

What you must turn in. For part 1, fill out the variables in the python template provided as per below. Comments are provided in the file itself explaining how you should fill it out, so please consult these first if you have any question as to how you should write up your answer.

`http://web.mit.edu/6.863/fall2012/writeups/assignment6/assignment6_template.py`

For part 2 we ask that you submit your report as a pdf file.

Once you have answered all questions and verified that your answers to part 1 are in the correct format by running the script (it will report any format errors or questions you still have to fill out), email your filled-in assignment6_template.py and assignment6_part2_report.pdf files to 6.863-graders@mit.edu, with the title "6.863 Assignment 6"

Part I

Statistical Parsing & Lexical Semantics

Initial Preparation:

If you haven't done so already, you should read chapter 14 on statistical parsing in your textbook, also available here:
`http://www.mit.edu/~6.863/fall2012/jmnew/ch14.pdf`

1 Using modern probabilistic parsers

1.1 Running statistical parsers on Athena

For this portion of the assignment, we've set up installations of several widely-used statistically-based parsers on Athena. Each is pre-trained on the Wall Street Journal (WSJ) section of the Penn Treebank (PTB), a collection of approximately one million words of running text from, ah, the *Wall Street Journal*, which was then converted, partially by hand, into parse trees.[1]

Such parsers require their input to be *tagged* – their rules do not go down to the level of individual words, only their parts of speech (POS). These part of speech names are partly a historical artifact, derived and elaborated from much older corpus work first done at Brown University in the 1960s. For example, the tag `IN` denotes any Preposition, while `VBD` stands for a past-tense verb, sometimes ending in *ed* or *en*, such as *taken* (but `VBD` could also be the tag associated with an irregular past tense verb such as *sung*). A list of 48 of the most important tags used for the Penn Tree Bank is given just below. You'll soon grow to know and love or hate this list. (For example, what information do these tags include? What information do they omit? You don't have to answer these questions, just think about them.)

`http://bulba.sdsu.edu/jeanette/thesis/PennTags.html`

Since the sentences don't come with their tags on them, and since the same word can have more than one tag, parsing systems must pre-process sentences and assign tags to each word.

We will be using these parsers with the Stanford maximum entropy tagger, which combines information about the endings of words plus some local context, such as the preceding word tag, and combines this information probabilistically to figure out the most likely tag to assign to the word it is currently looking at. For now if you want you can download an alternative standalone maximum entropy tagger implementation (MXPOST) yourself and read about it here:

`http://www.inf.ed.ac.uk/resources/nlp/local_doc/MXPOST.html`

and read about the method via the following paper as described there:

`http://www.mit.edu/~6.863/fall2012/readings/mxpost_doc.pdf`

1.1.1 Linguist Parser Setup

For a warm-up introduction to these parsers, we will use a java-based interface "Linguist" (LINguistic Graphical User Interface and Syntactic Toolkit) that lets you switch between the Bikel-Collins, Stanford, and Berkeley statistical parsers. It lets you browse the entire training set of Wall Street Journal sentences and evaluate the parsers on the designated *test* set of Wall Street Journal sentences. The toolkit has other useful functionality that we will run through as we go.

If you're at an Athena cluster, it's pre-installed in /mit/6.863/fall2012/code/assignment6/linguist so to run it just copy it over to your home directory and run:

```
add 6.863
cp -r /mit/6.863/fall2012/code/assignment6/linguist ~/linguist
cd ~/linguist
./run.cmd
```

If you are not at an Athena cluster and would like to run the program on Athena remotely, you must use `ssh` X-windows forwarding option (note this will be slow, it will not work on Windows, and will not work on Mac OS X unless you have installed X11; if you encounter issues, go to an Athena cluster):

```
ssh -Y mylogin@athena.dialup.mit.edu
```

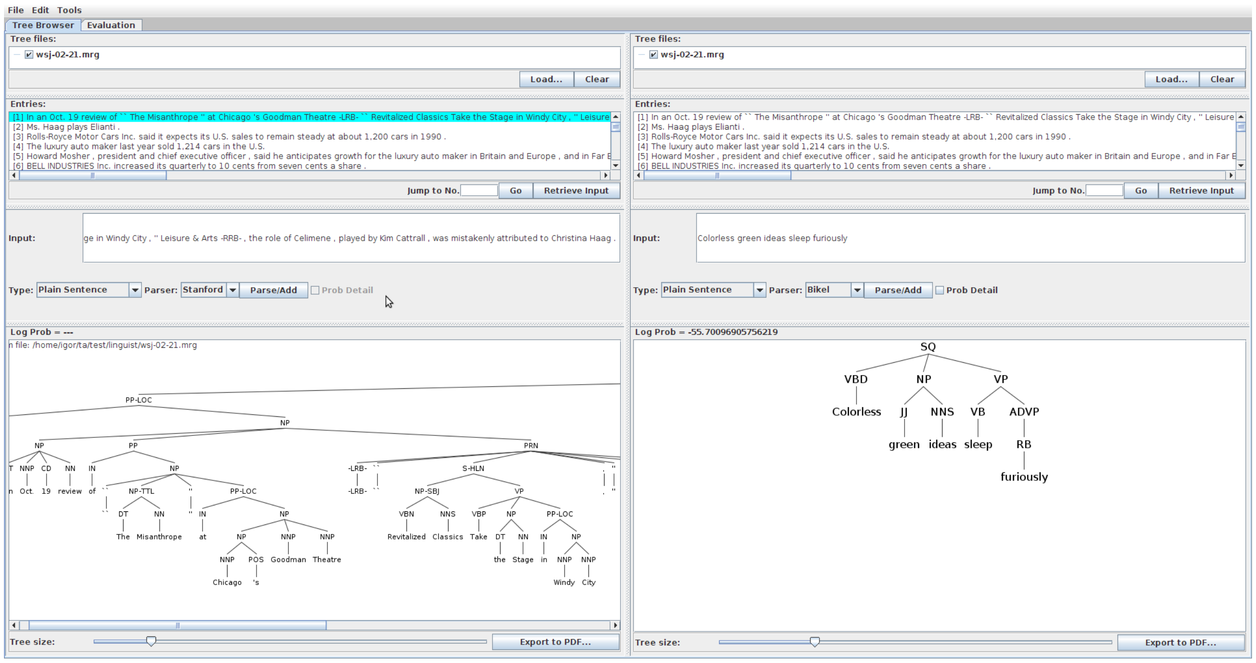
[1] As is standard, specifically this means the systems are trained on sections 02 to 21 of the Wall Street Journal, approximately 40,000 parsed sentences.

Alternatively, this parser wrapper comes packaged as a `.jar` file `linguist.jar` so it should be possible to install and run on your local linux, Windows, or Mac OS X machine (that said, it's pre-installed on Athena, so if you run into issues just go to an Athena cluster). If you are installing on your own machine, download the installation files and read the `linguist-install.txt` file for setup instructions:

```
http://web.mit.edu/~6.863/fall2012/code/assignment6/linguist-install.txt
http://web.mit.edu/6.863/fall2012/code/assignment6/linguist.zip
```

1.1.2 Linguist Parser Usage

Let's give a brief overview of what the layout of this gui, and what you can do with it. There are two basic modes that will be used in the assignment: the Tree Browser and the Evaluation mode, which can be switched to back and forth by using the relevant tabs at the top of the window.



wsj-02-21.mrg

[1] In an Oct. 19 review of `` The Misanthrope '' at Chicago 's Goodman Theatre -LRB- `` Revitalized Classics Take the Stage in Windy City , '' Leisure
[2] Ms. Haag plays Elianti .
[3] Rolls-Royce Motor Cars Inc. said it expects its U.S. sales to remain steady at about 1,200 cars in 1990 .
[4] The luxury auto maker last year sold 1,214 cars in the U.S.
[5] Howard Mosher , president and chief executive officer , said he anticipates growth for the luxury auto maker in Britain and Europe , and in Far E
[6] BELL INDUSTRIES Inc. increased its quarterly to 10 cents from seven cents a share .

Input: ge in Windy City , '' Leisure & Arts -RRB- , the role of Celimene , played by Kim Cattrall , was mistakenly attributed to Christina Haag .

Type: Plain Sentence Parser: Stanford

Log Prob = ---

n file: /home/igor/ta/test/linguist/wsj-02-21.mrg

Input: Colorless green ideas sleep furiously

Type: Plain Sentence Parser: Bikel

Log Prob = -55.70096905756219

Browser Mode The Browser mode is default. In this mode you can load a corpus and display the parse trees. You can also enter custom inputs in the form of a plain sentence, a tagged and tokenized sentence, and a parenthesized s-expression corresponding to the full parse of the sentence. The plain text and tagged sentence inputs can then be parsed using any one of the three available parsers.
Here's how the gui layout divides up. First note that it is split into left and right-hand sides which encompass parallel functionality to be able to visualize tree pairs side-by-side.

Top to bottom, there are roughly four regions to note:

1. **File Input** An area that displays the list of active corpus files. Click the "Load..." button to open up a corpus file. The corpus is a plain-text file containing s-expressions corresponding to sentence parses. You can load multiple files at a time by selecting several files. Alternatively you can load a directory with file filters. You can reset the corpus files by clicking "clear".

2. **Treebank sentences** An area that displays sentences loaded into the system. Initially, empty, it will be populated with sentences from the corpus as soon as soon as you load the files. By selecting a particular sentence item you will be able to display a visual representation of the parse in the bottom panel.

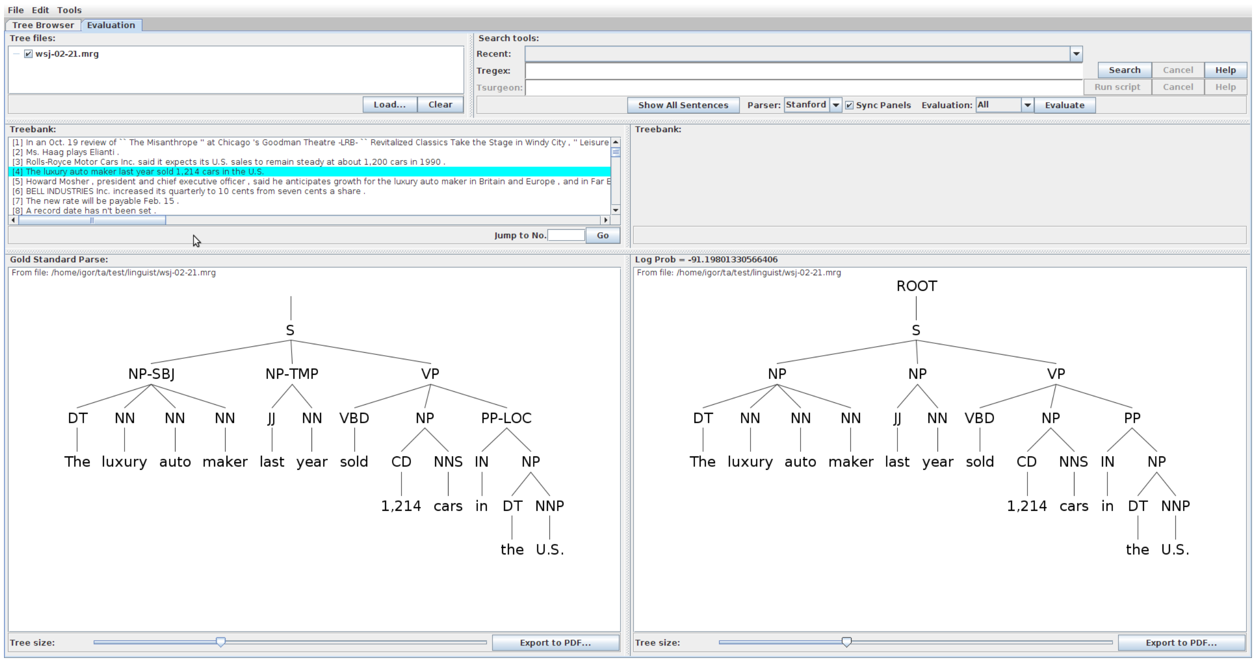
3. **Custom Input Box** Next, there is a box that lets you enter custom input to be parsed. There are three possible input types which you taggle using the drop down box immediately below the input box: Plain Sentence, Tagged Sentence, and S-exp (no parsing).

   A tagged sentence input needs to be in s-expression format: `(token (TAG))`. If you entered the input of type plain or tagged sentence, also select a parser model to be used in parsing – and press the "parse/add" button. This will add the sentence to the list of entries and display the parse tree below. You can also add fully specified s-expression tree inputs.

4. **Log Probability Scoring**

   The log probability score will automatically be displayed for each parse. If you select the checkbox "Prob detail" and use the Bikel-Collins parser to parse a custom input sentence, a new window will pop up that will display log prob values for each sub-portion of the tree. This feature will be described at greater depth in Part II.

5. **Parse tree output:** The region below is where the parse tree output is displayed. These are scrollable, with zoom-in and zoom-out scroll-bar. The "export to pdf" button will bring up a menu selection for outputting the parse to a pdf file.



Tree files: wsj-02-21.mrg

Search tools:
Recent:
Tregex:
Tsurgeon:

Parser: Stanford ☑ Sync Panels Evaluation: All

Treebank:
[1] In an Oct. 19 review of `` The Misanthrope '' at Chicago 's Goodman Theatre -LRB- `` Revitalized Classics Take the Stage in Windy City , '' Leisure
[2] Ms. Haag plays Elianti .
[3] Rolls-Royce Motor Cars Inc. said it expects its U.S. sales to remain steady at about 1,200 cars in 1990 .
[4] The luxury auto maker last year sold 1,214 cars in the U.S.
[5] Howard Mosher , president and chief executive officer , said he anticipates growth for the luxury auto maker in Britain and Europe , and in Far E
[6] BELL INDUSTRIES Inc. increased its quarterly to 10 cents from seven cents a share .
[7] The new rate will be payable Feb. 15 .
[8] A record date has n't been set .

Jump to No.

Gold Standard Parse:
From file: /home/igor/ta/test/linguist/wsj-02-21.mrg

Log Prob = -91.19801330566406
From file: /home/igor/ta/test/linguist/wsj-02-21.mrg

Evaluation Mode In this mode you can execute structured queries over a corpus and evaluate the different parsing models with reference to a gold standard corpus.

1. **File Input** Similarly to the browser mode, this area displays the list of active corpus files. The principal difference is that in this mode you can only load one corpus (the gold standard) at a time.

2. **Search Tools** To the right of the file input list you will see a set of boxes and buttons for executing structured queries over the corpus. The input is provided in a tree regular expression format ("tregex") which is described in a pop-up window if you press the "Help" button on the right. This panel also includes the ability to execute tree transformations in conjunction with search. The option is beyond the scope of this assignment and is disabled by default, but can be enabled if you go to "Tools ⇒ Options" and select the "Enable Tsurgeon" checkbox.

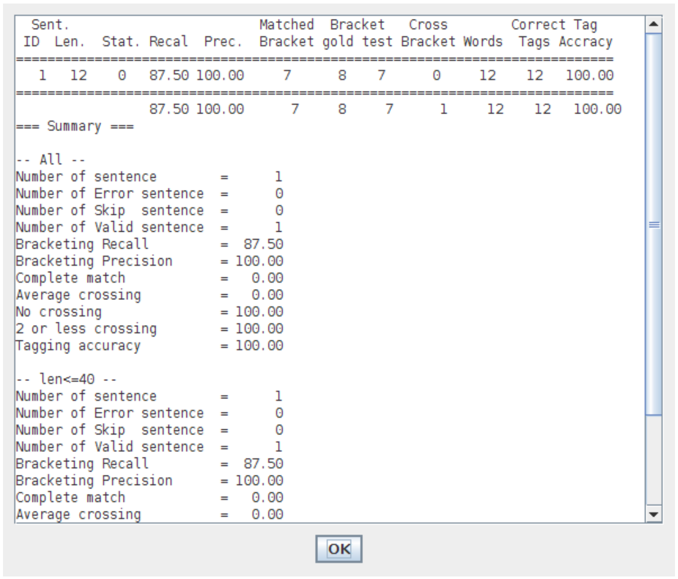
3. **Treebank sentences** There are two areas for displaying the treebank sentences. The left hand side panel is designated for the gold standard parses loaded from file. The right hand side panel displays the same set of sentences parsed by a specified parser. If the synchronized option is checked below, then the right-hand-side is disabled, and the parsing is synchronized on the sentence currently highlighted on the left-hand-side.

4. **Evalb scoring**

   Directly below the search tool panel is a set of options for evalb scoring. Selecting a parser type from the drop-down box , toggling to the "selected" drop-down menu option and clicking the button "Evaluate" will spawn a window that displays the results of running the precision/recall `evalb` program using the gold tree against the parser's output. The most important numbers here are the Recall and Precision values, along with the negative log probability score for the parse tree (on this scale, closer to 0 is a higher probability, with logs taken to base e).

   If you select the "All" drop down menu option, the entire active corpus will be parsed by the specified parser and evaluated. Note that if the corpus exceeds 100 sentences this might take some time to run to completion.

To see an example of what evalb scoring analysis looks like for a parse, in the snapshot below we have selected the fourth sentence, *The luxury auto maker last year sold 1,214 cars in the U.S.* ., as parsed by the Bikel system. You can see the gold standard parse on the left, and what the parser produces on the right. Evalb returns 87.50% recall and 100% precision values. The log probability score is approximately −53.65.



```
  Sent.                        Matched  Bracket   Cross        Correct Tag
 ID  Len.  Stat. Recal  Prec.  Bracket gold test Bracket Words  Tags Accracy
============================================================================
   1   12     0   87.50 100.00      7      8    7      0     12     12  100.00
============================================================================
                  87.50 100.00      7      8    7      1     12     12  100.00
=== Summary ===

-- All --
Number of sentence        =      1
Number of Error sentence  =      0
Number of Skip  sentence  =      0
Number of Valid sentence  =      1
Bracketing Recall         =  87.50
Bracketing Precision      = 100.00
Complete match            =   0.00
Average crossing          =   0.00
No crossing               = 100.00
2 or less crossing        = 100.00
Tagging accuracy          = 100.00

-- len<=40 --
Number of sentence        =      1
Number of Error sentence  =      0
Number of Skip  sentence  =      0
Number of Valid sentence  =      1
Bracketing Recall         =  87.50
Bracketing Precision      = 100.00
Complete match            =   0.00
Average crossing          =   0.00
```

1.2 Basic questions about the statistical parsers

Problem 1.1. First, a question about tagging. The simple NLTK parser you used before in Assignment 5 was also assigning parts of speech to words, but it was doing it using the rules it learned such as:

```
VBN -> 'classified' [p=0.0004766]
```

This could be considered a kind of *unigram tagger*, because it does not use any surrounding context to assign parts of speech. In the Stanford/Bikel/Berkeley parsers' maximum-entropy tagger, on the other hand, use surrounding context to make part of speech decisions. But there is a bit of information lost at the boundary between the maximum entropy tagger and the Bikel parser.

What information was present in the NLTK parser's part-of-speech assignments that is *not* conveyed between the maximum entropy tagger and the Bikel parser?

We would expect the trained parser to do well on the sentences it has already seen — the sentences trained on. But does it always do perfectly on training data and return a parse tree that is identical to the ones it was trained on? Using the java-based parser, select the **Evaluation** tab, load the corpus **wsj-02-21.mrg** (located at **/mit/6.863/fall2012/code/assignment6/wsj-02-21.mrg**), check the box labeled, "Sync Panels," as well as the drop down option "Bikel" next to the "Parser" label. Now you can step through each sentence of the training data, one by one, to see how the parser does on them. As you select each one, the system will automatically tag and then parse that sentence. The left-hand panel will display the original training tree, while the right-hand panel will show the parser's output. The very first sentence is this one, 49 tokens long:

```
In an Oct. 19 review of `` The Misanthrope '' at Chicago 's Goodman
Theatre -LRB- ``Revitalized Classics Take the Stage in Windy City , ''
Leisure & Arts -RRB- , the role of Celimene , played by Kim Cattrall ,
was mistakenly attributed to Christina Haag .
```

The parser does perfectly on this first training example and its output tree structure matches the training data. (We put to one side some of the tag details in the original training data for nonterminals like `NP-TTL`, which denotes an NP that is a 'title' that are not meant to be learned by the parser; this is also true of any 'empty' nodes in the training data trees that dominate null words, labeled with a 0 – see training sentence number 5 for an example.) Note that if you click the "Evaluate" button, checking the "Selected" Option in the drop down menu immediately to the left, you will see a pop-up window, which shows the results from the standard precision/recall `evalb` program.

```
Ms. Haag plays Elianti .
```

Problem 1.2. What is the first sentence in the training set where the (Bikel) parser makes an error with Precision < 100%? Report the sentence number and the actual sentence. (You can find it by going down the list of training sentences, and as you select each one, press the "Evaluate" button. Look for the first sentence where the Precision value is less than 100.00%.) How would you describe the kind of error that the parser makes on this sentence? Does this error have any effect on the meaning of the sentence? That is, would the error the parser makes have an effect on the semantic interpretation of the sentence, as it does in cases where a parser can decide to attach a phrase at one place versus another, as in *I saw the guy on the hill with the telescope*, where *with the telescope* might be part of the NP *the hill* or part of the NP *the guy* or the VP *saw*. Explain briefly.

2 Current issues for statistical parsers

2.1 Assumptions

Statistical parsers must make statistical assumptions. As you have seen from the parsers' output, one of the assumptions is that we can measure the likelihood of sentences by a negative log score. The more negative the score, the less likely the sentence.

Problem 2.1. This problem asks you to consider some simple properties of this score. To examine this, use the Bikel parser and sentences following a pattern like this, where we have used braces and a | following a regular expression notation to indicate "choose one of these words":

The {witch | broker} is dead .

The wicked {witch | broker } is dead .

The wicked wicked {witch | broker } is dead .

The wicked wicked wicked {witch | broker} is dead .

The wicked wicked wicked wicked {witch | broker} is dead .

In the Browser mode, using the Bikel parser, analyze these sentences and look at their log probability scores. **What is the relationship between sentence length and sentence likelihood, all other things being equal? Is there any difference between the scores when you substitute *broker* for *witch*? Why or why not?**

Problem 2.2. Some researchers have suggested using the log probability score as a substitute for that awkward notion of "grammaticality" that linguists like to use. The idea is that less grammatical sentences should have a lower likelihood. (Hmm... something like in your Reading and Response 1.) Now we have a parser to test this idea. Keeping in mind your answer from the previous problem, and the "all other things being equal" caveat, below are a set of sentences. Those marked with asterisks are generally considered as "ungrammatical," and therefore highly unlikely, to varying degrees. (Let's accept this as a given.) Your job is to run these sentences through the parser, and examine the log probability scores to see how well the notion of "grammaticality" aligns with that of "low likelihood," at least for these examples. (You might also have to examine the parse treesto make sure the parser is not going wildly astray.) Note that you also may have to construct new, similar examples to adjust for what you discovered in the previous question. **List the log probability of the parses of each of these sentences (using the Bikel parser). Based on this, do you think the alignment between grammaticality and likelihood is accurate?**

(1) a. *John slept .*
b. * *John slept Bill .*
c. *Bill was arrested .*
d. * *It was arrested Bill .*
e. *I tried to leave .*
f. * *I tried John to leave .*
g. *I promised John to leave .*

2.2 How well do lexicalized parsers work?

If you haven't already done so, please read the JM chapter 14 assignment mentioned at the start of the assignment, and then come back here when you are done. OK, if you've read that chapter, you'll recall that a key advance claimed for these state-of-the-art statistical parsers is that they use information about the *head* of a phrase, encoded as part of the nonterminal information in context-free rules. This allows a parser to take into account distinctions between different words and condition the parse on them, what are called *lexical dependencies.* For example, the *head* of a verb phrase (VP) is its verb, the head of a noun phrase (NP) is[2] its noun, and the head of a prepositional phrase (PP) is its preposition. What the rest of the phrase looks like often greatly depends on which head this is. For example, if it is the verb *saw*, then we would expect that a Noun Phrase (NP) follows (as in *saw the guy*), but if the verb is *sleep*, then the parser should not expect an NP to follow. (We can think of the verb as a function taking some number of arguments.) In fact, since the NP that follows will itself have its own "head", e.g., *guy*, the parser can take that information into account as well. This kind of information is often helpful in disambiguating examples such as the following (from the JM textbook):

```
Workers dumped sacks into a bin
```

[2]On some accounts. There is a good argument that the head of an NP is in fact the Determiner, or Det, but we put this aside here.

Here, the Prepositional Phrase, *into a bin*, with a head *into* might modify either the VP with the head verb *dumped*, specifying where the sacks were dumped; or an NP with a head noun *sacks*, specifying the type of sacks that are being dumped. The first interpretation, revealing a closer "affinity" between *dump* and *into* rather than *sacks* and *into.* In short, we must condition the parse on the precise words that occur – in particular, the heads of the phrases involved. We do this by adding the head information to the individual rules expanding phrases (details given in Chapter 14 of JM), e.g., instead of "bare" phrase structure rules with probabilities:

```
VP -> VBD NP    0.8
VP -> VBD NP PP 0.2
NP -> NNS       0.8
NP -> NNS PP    0.2
```

We have instead the rules annotated with their corresponding heads, indicating the affinity between particular verb, noun, and prepositional heads directly (here we have made up the probabilities to show that we have 'boosted' the probability of having the PP with *into* tied more closely to *dumped* than with *sacks*):

```
VP-dumped -> VBD-dumped NP-sacks         0.4
VP-dumped -> VBD-dumped NP-sacks PP-into 0.6
NP-sacks -> NNS-sacks PP-into 0.4
NP-sacks -> NNS 0.6
```

The JM book uses this notation:

```
VP(dumped) -> VBD(dumped) NP(sacks) PP(into)
```

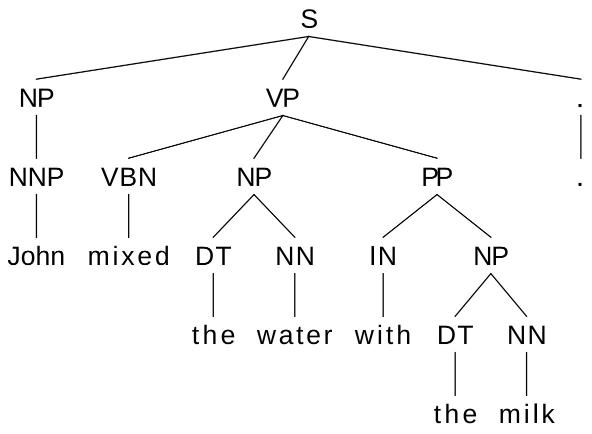
Of course, we now have a great many more rules, as noted in lecture: assuming binary branching rules, if there are N nonterminals, and we have an alphabet Σ, then since we can have any one of 3 possible nonterminal choices in a rule, and two alphabet (word) symbols, one for the head, and one for the non-head, then there are $|N|^3 \times |\Sigma|^2$ possible rules, a huge number. Since the CKY algorithm runs in time at worst proportional to the cube of the length of the sentence, n^3, times the grammar size, this not such good news! Fortunately, the number of distinct words in a sentence of length n can be at most n, so we can reduce this grammar size factor to $|N|^3 \times n^3$, for a worst case running time of $O(n^5|N|^3)$. That is still not terrific.

Further, we have many more parameters to estimate. In practice, the particular "affinity probabilities" are estimated by examining their frequencies of occurrence in an actual corpus — a count of how many times we actually see a PP with the head *into* related to a VP with the head *dumped* as opposed to appearing after an NP with the head *sacks.* As you can imagine, since there are many, many different nouns and verbs, this leads to a challenging estimating problem. Even in a corpus of 49,000 sentences, we would rarely expect to see a particular verb appear in the same context with a particular noun or preposition. Thus, as discussed in JM, what such systems must do is (1) break the total conditional probability chain down into simpler Markov-like parts; and (2) employ some method of smoothing or back-off for missing data. For example, in the Bikel-Collins parser, given some particular verb, e.g., *dumped*, if a specific head noun right context is not found, e.g., *dumped–sacks*, then the system will first back-off to use just the tag of the head, NNS, and if there is no estimate in the corpus for that conditioning value, then a vanilla un-lexicalized context-free rule will be used.

Let's take a quick look at how well this works in practice to resolve the Prepositional Phrase attachment ambiguities that make natural language processing so challenging.

Problem 2.3. We will consider just a single kind of sentence, of this general sort:

(2) *John mixed the water with the milk .*



The Bikel parse for this is as above. Note that the Prepositional Phrase (PP) *with the milk* is attached "high," that is, directly under the VP, rather than "low," as part of the Noun Phrase *the water*. This high attachment means that *the milk* is considered to modify the mixing event, rather than being part of the water, and we are mixing both together. This is probably a sensible result. On the other hand, if we had a sentence such as, *John ate the ice-cream with vanilla* then it seems more sensible to think of the vanilla as being part of the ice-cream. In this problem we'll see how subtle this relationship can be, and whether the statistical parsers successfully "learn" about it from their training data.

Let's see what happens when we interchange *water* and *milk*, as well as try other combinations of verbs and nouns. That is, parse the following sentence and see whether the PP *with water* is attached "high" or "low" by examining the resulting parse tree.

(3) *John mixed the milk with the water .*

In fact, we should be a bit more systematic: try the same experiment and see whether attachment is HIGH or LOW by constructing all the variants of this sentence type with the verbs *mixed* and *sold*, and the nouns *milk, water,* and *stock*. We've begun the list below. Your job is to specify whether the PP *with the ...* is attached "HIGH" or "LOW" for each case. We have placed HIGH into the table as your first entry, as per the results from our example *mixed the water with the milk*. You are to fill out the rest of the table, with either the entries HIGH OR LOW, first recording the results from your parse of *mixed the milk with the water*, and then the rest of the table entries, which are blank.

Verb	Noun	Noun with milk	milk with Noun
mixed	water	HIGH	???
mixed	stock		
sold	water		
sold	stock		

Problem 2.4. What explains the pattern of HIGH and LOW attachments that you got in the previous problem? To begin to investigate this, we should think about why there should be any differences between the verbs and nouns you select and their attachment preferences – that is, how do such systems learn any preferences in the first place? See if you can figure this out by examining the training corpus file `wsj-02-21.mrg` (yes, it's a large file, but you can use emacs, vi, less or tregex[3] to search through it): look for sentences that contain *milk* as an object, and see whether they also have PPs that modify *milk* or not, as compared to the other verbs and nouns above. Please explain in a few sentences what evidence you can find for why the Bikel parsing system follows the pattern of HIGH and LOW attachments that it does for the cases in your table. What does this imply about the nature of the training data and learning from corpora?

[3]You can read about tregex (tree regular expressions) in part I of this assignment.

This concludes Part I!

Part II

You have seen in lectures that one of the goals of statistical parsing methods has been to resolve ambiguity by using training examples to select which of several alternatives is most likely. To do this, they have incorporated lexical information in the form of, for example, the 'heads' of phrases projected up from lexical entries. For instance, in a Verb Phrase (VP) form like *eat ice-cream*, the verb *eat* is in effect copied up to the VP so that it can play a role in conditioning the expansion of the VP, as to whether, e.g., an NP with a particular head can follow or not.

Such a method seems to capture at least the **syntactic** regularities that accompany different verb types, but the question remains as to whether **semantic** regularities are captured this way. By 'semantic regularities' we mean regularities that are reflected in possible/impossible 'alternations' as discussed in Beth Levin's *English Verb Classes and Their Alternations* (1973). For example, we have the following differences between pour and fill:

(4) *John filled the glass with water.*
(5) * *John poured the glass with water.*
(6) *John poured water into the glass.*
(7) * *John filled water into the glass.*

Such examples are called **verb alternations**. Since these differences are reflected in alternative Prepositional Phrases that are possible or impossible with *fill vs. pour*, one might at least hope that such differences might be picked up by training over some set of examples like the PTB, but of course there are issues as to the sparseness of the training data. And even if such data are present, there is a question as to whether such parsers 'learn' the preferences that people (i.e., you) seem to have about such examples.

To make this all concrete, we will assign each of you **one** particular verb from Levin's examples. Everybody will have a verb of their own. You will also be able to access in pdf the relevant section of Levin's book that pertains to the alternation analysis of these two verbs, providing you with Levin's examples and her analysis of what verb subcategorization frames ought to appear with such verbs. Throughout the course of this part of the Assignment you will collect and analyze four aspects of your verb:

1. The frequency with which the verb appears in the PTB, along with its associated syntactic argument frames (e.g., NP, NP PP, etc.);
2. Levin's analysis of what frames ought to appear with the verb, according to semantic regularities, along with her examples;
3. Your intuitions as to which verb subcategorizations are more likely than others for your verb;
4. How the Bikel-Collins parser ranks each possible subcategorization frame (using negative log probability scores), according to the parsing of test sentences that you construct for your verbs.

Using your findings from 1–4, above, in your Assignment report we want you to address how the syntactic and semantic regularities line up, comparing Levin, your intuitions, the PTB, and the Bikel-Collins parser.

We will use the verb *abandon* in what follows as a running example so you can see explicitly the workflow of what you must do and the questions you must address.

Required reading, software, and data

You can do this lab most easily at an Athena machine; refer to part 1 for setup instructions.

Reading:

1. The Levin book, *English Verb Classes and Alternations* (EVCA), 1993. This is available in 3 sections, in pdf format, covering the entire book (though you'll only need to look at parts), as follows, where the numbering refers to sections in the Levin book:
   `http://www.mit.edu/~6.863/fall2012/readings/levin1-5.pdf`
   `http://www.mit.edu/~6.863/fall2012/readings/levin6-29.pdf`
   `http://www.mit.edu/~6.863/fall2012/readings/levin6-29.pdf`
   `http://www.mit.edu/~6.863/fall2012/readings/levin30-57.pdf`

2. The index for the Levin book, which cross-references verbs by section numbers,
   `http://www.mit.edu/~6.863/fall2012/readings/evca93.index`

3. The sections 2-21 of the Penn TreeBank in its 'merged' form used for training the Bikel-Collins parser. The file `wsj-02-21.mrg` is available from `/mit/6.863/fall2012/code/assignment6/wsj-02-21.mrg`.[4]

Now on to your particular tasks. You have been assigned a verb to work with; to see your verb, check

`http://web.mit.edu/6.863/www/fall2012/labs/assignment6_levin_verb_assignments.txt`

(if your name isn't in that list, email the TA)

Collecting and analyzing subcategorization frames

First, we want you to compare the occurrences of your verb in particular syntactic frames in the PTB with the descriptions of the occurrences of this verb in Levin (1993) (i.e., your copy of the section of her book for your verb). This task has the following sub-parts:

1. **Retrieve all the 'subcategorization frame' occurrences of your verb in the Penn Tree Bank, by using the `tregex` search tool in the Linguist interface.**

2. **Compare these retrieved contexts to Levin's possibilities for your verb and note any differences. Are there contexts that don't occur in the PTB but do occur in Levin, and vice versa? Tabulate the # of occurrences of each frame in the PTB, e.g., VP NP vs. VP NP PP, and the type of each PP (locative, temporal, etc.), noting any semantic differences from what Levin says should occur.**

Let's now go through this part of the exercise with our concrete example, *abandon*.

Locating all verb occurrences

To find all the occurrences of *abandon* (including basic inflections) as a verb, we can execute a `tregex`[5] query in the following way. First start Linguist, go to the Evaluation tab, and Load the Penn Tree Bank from /mit/6.863/fall2012/code/assignment6/wsj-02-21.mrg

Then enter the following expression in the Tregex search box (of course, your query will be different based on the verb you were assigned), and click "Search" (if you see lots of error messages or you don't see anything happen after 30 seconds, restart the program and make sure you typed in the query correctly).

[4]This corpus is licensed to MIT for academic use only. Please do not distribute.

[5]You can read about tregex (tree regular expressions) in part I of this assignment.

```
@VP < ( __< /^[Aa]bandon[esi]|^[Aa]bandon$/ )
```

There are additional options that you will need to set. In order to extract the subcategorization frame, we just need the matched subtree. Also, certain trees may have multiple matches – all of which should be taken into account. To be able to display these you need check navigate to Tools ⇒ Options and make sure that the box labeled "Show only matched portions of the tree" is checked.

After executing the query, you can save the results of the query in a file, in the form of just the word sequence or the s-expression corresponding to the subtree by selecting "File ⇒ Save matched gold std trees" or "File ⇒ Save matched sentences".

The regular expression that follows searches for a string that is dominated by anything, signified by the wild card symbol `__` and then dominated (<) by a VP. In turn, this wild card symbol matches any label that is immediately dominated by a VP node, hence the < after the VP node.

Instead of the wild card symbol, the VB symbol could also be used. Each verb has a category label beginning with VB, following the Penn Treebank tags format.[6]

However, use of the wild card symbol is advantageous in that it can match incorrectly labeled verbs. For example, `tregex` matched an instance of *awe* that was marked with the label `JJ` to indicate an adjective. Yet this label was directly dominated by a VP node. For example, inspection of the sentence, *Ms. Johnson, who works out of Aetna' s office in Walnut Creek, an East Bay suburb, is awed by the earthquake's destructive force* reveals that despite the labeling as an adjective, *awe* is used as a verb in this construction. If `tregex` searched for a VB labeled target item, it would not match *awe* in this construction.

In order to match a particular target verb, we use a regular expression. For *abandon*, this is the regular expression:

```
 /^[aA]bandon[esi]|^[aA]bandon$/
```

The `abandon[esi]` pattern will match any inflected forms of the verb abandon, such as abandoned, abandons, or abandoning. To match the uninflected form of abandon we use the pattern ^*abandon*$. For additional information on `tregex`, you can refer to the help file by clicking the help file:

Executing `tregex` using the command args above we get a list of matching PTB sentences. If you get lots of exceptions upon executing the search, make sure that you typed the search expression correctly, and that you wait a few seconds between when you load the corpus file and execute the query (the software has some race conditions).

Collecting subcategorization frames

The format of a subcategorization frame is typically: `[XP X args]`. So for the first match above, the extraction program should output: `[VP (VB abandon) NP]`. You may use the provided script: `frame_extractor.py` (located at /mit/6.863/fall2012/code/assignment6/frame_extractor.py) to help you extract the frames, or you can roll your own. You can use frame_extractor.py as follows (where abandon.par is what you get by saving the matched portions of gold std trees; make sure that "Show only matched portions of the tree" in Tools ⇒ Options is checked when saving the gold std trees)

```
python frame_extractor.py -p abandon.par
[VP (VB abandon) NP]
[VP (VBN abandoned) NP PP PP-TMP]
[VP (VBN abandoned) NP PP PP-TMP]
... (etc) ...
```

[6]For reference on the POS labels, please see the Treebank page: `http://web.mit.edu/6.863/www/PennTreebankTags.html`

For additional context, you can print out the verb arguments as a string (if using frame_extractor.py, you can do this by adding the argument **-x true**). You will need to analyze the contents of verb arguments to check whether they agree with Levin. (For example, to determine whether the argument is a location or an event.)

You should include in your report a tabulation of subcategorization frame counts like in Table 1 below. (You should ignore case when counting).

	subcategory	count
a)	[VP (VB abandon) NP]	15
b)	[VP (VBD abandoned) NP]	12
c)	[VP (VBN abandoned) NP]	9
d)	[VP (VBG abandoning) NP]	7
e)	[VP (VBN abandoned)]	4
f)	[VP (VBZ abandons) NP]	2
g)	[VP (VBD abandoned)]	1
h)	[VP (VBN abandoned) PP-CLR]	1
i)	[VP (VBN abandoned) PP]	1
j)	[VP (VBP abandon) PP]	1
k)	[VP (VB abandon) NP PP-CLR]	1
l)	[VP (VB abandon) NP S-CLR]	1

Table 1: Table of Subcategorization Frames for abandon

Now, compare the tabulated results to what Levin says. According to Levin, *abandon* is a verb that patterns with *leave* and occurs in the frame [VP NP] where the direct object is the location that has been left.

[V NP] as in, *We abandoned the area* (Levin, p. 264)

In examples (a,b,c,d,f), some form of the verb abandon occurs in a verb frame with a direct object NP alone. A look at these sentences also shows that the direct object of *abandon* in the Penn Treebank constructions is **not** a location. For example, in sentence 1979, the direct object of *abandon* is *the provision* and in sentence 3191 the direct object is *the president's proposal for a cap on utilities' sulfur-dioxide emissions.* The other verb frames, (e,g,h,i,j,k,l), that appear in the PTB with abandon in are not discussed in Levin's work.

Now you are to do the same analysis for your assigned verb. Please write up your findings and along with relevant supporting data. Pay special attention to any semantic 'divergences' as with the example of *abandon* – Levin says that this will be a location, but this is not usually true in the PTB.

This concludes Assignment 6